Package 'EJAM'

January 3, 2024

Title EJAM Environmental Justice Analysis Multisite tool

Version 2.2.1

Author Mark Corrales

Maintainer Mark Corrales <corrales.mark@epa.gov>

License MIT + file LICENSE.md

Description Tools for summarizing environmental and demographic indicators
(such as those in EJScreen) for residents living near any one of a number of
specific sites. It uses quad tree search/indexing of block locations, data.table, etc.
to provide very fast identification of nearby blocks, distances, and
aggregation of indicators within each distance. It can be uses as a web app, with the user interface provided by R shiny.

URL https://github.com/USEPA/EJAM

Depends R (>= 2.10),
aws.s3,
EJAMbatch.summarizer,
EJAMejscreenapi,
shinyBS

Imports attempt,
collapse,
config (>= 0.3.1),
data.table,
DBI,
doSNOW,
dplyr,
DT,
foreach,
ggplot2,
ggridges,
glue,
golem (>= 0.3.3),
hrbrthemes,
htmltools,
leaflet,
magrittr,
methods,
openxlsx,
pdist,

pkgload,
readxl,
rhandsontable,
rmarkdown,
RMySQL,
SearchTrees,
shinydisconnect,
sf,
shiny (>= 1.7.2),
shinycssloaders,
shinyjs,
sp,
tidyr,
tidyverse,
viridis

Suggests knitr,
spelling,
testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Language en-US

VignetteBuilder knitr

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Config/testthat/parallel true

R topics documented:

`.onAttach` *.onAttach - Do slow initialization steps - Download data, load key data into RAM, create index to all US blocks Note this duplicates some code in global.R, and see source code here to adjust settings.*

Description

.onAttach - Do slow initialization steps - Download data, load key data into RAM, create index to all US blocks Note this duplicates some code in global.R, and see source code here to adjust settings.

Usage

```
.onAttach(libname, pkgname)
```

Arguments

`libname`	na
`pkgname`	na

Details

Does this even happen if connect server runs app as a regular shiny app without loading all? In what order? see app.R too ***

all_equal_functions *all_equal_functions - UTILITY - check different versions of function with same name in 2 packages used by dupenames() to check different versions of function with same name in 2 packages*

Description

all_equal_functions - UTILITY - check different versions of function with same name in 2 packages used by dupenames() to check different versions of function with same name in 2 packages

Usage

```
all_equal_functions(
  fun = "latlon_infer",
  package1 = "EJAM",
  package2 = "EJAMejscreenapi"
)
```

Arguments

fun	quoted name of function, like "latlon_infer"
package1	quoted name of package, like "EJAM"
package2	quoted name of package, like "EJAMejscreenapi"

Value

TRUE or FALSE

See Also

dupenames() all.equal.function()

app_run_EJAM *app_run_EJAM - identical to run_app(), just an alias*

Description

launch Shiny web app from RStudio

Usage

```
app_run_EJAM(
  onStart = NULL,
  options = list(),
  enableBookmarking = "server",
  uiPattern = "/",
  ...
)
```

app_server *app_server - EJAM app server*

Description

app_server - EJAM app server

Usage

```
app_server(input, output, session)
```

Arguments

`input, output, session`
: Internal parameters for shiny. DO NOT REMOVE.

avg.in.us *avg.in.us (DATA) national averages of key indicators, for convenience*

Description

also available via usastats and created by /data-raw/datacreate_avg.in.us

bgej *bgej (DATA) EJScreen EJ Indexes for Census block groups*

Description

bgej (DATA) EJScreen EJ Indexes for Census block groups

Details

- As of 08/2023 it was the EJScreen 2.2 version of data, which used ACS 2017-2021.

Each year this should be re-created as for the latest version.

See https://www.epa.gov/ejscreen

bgpts *bgpts (DATA) lat lon of popwtd center of blockgroup, and count of blocks per block group*

Description

This is just a list of US block groups and how many blocks are in each... It also has the lat and lon roughly of each blockgroup

Details

The point used for each bg is the Census 2020 population weighted mean of the blocks' internal points. It gives an approximation of where people live and where each bg is, which is useful for some situations.

```
  As of 8/2023 it is the EJScreen 2.2 version of data, which uses ACS 2017-2021
  and Census 2020. it has all US States, DC, PR, but not  "AS" "GU" "MP" "VI"

  How lat lon were estimated:

# Now, for Census 2020 blocks, create pop wtd centroids lat lon for each block group ####
#  using  blockwts and   blockpoints

bgpts_blocks <- copy(blockpoints) # not essential but ok to make sure we do not change blockpoints it
# all.equal(bgpts$blockid , blockwts$blockid)
bgpts_blocks[ , bgid    := blockwts$bgid]
bgpts_blocks[ , blockwt := blockwts$blockwt]
# get pop wtd mean of lat, and same for lon, by bgid
bgpts <- bgpts_blocks[ , lapply(.SD, FUN = function(x) stats::weighted.mean(x, w = blockwt, na.rm =
rm( bgpts_blocks)
# add the bgfips column, so it has bgfips, bgid, lat, lon
# all.equal(bgpts$bgid,bgid2fips$bgid)
bgpts[ , bgfips := bgid2fips$bgfips]
# setnames(bgpts, 'bgfips', 'FIPS')

# BUT NOTE this census2020 block table has PR but lacks "AS" "GU" "MP" "VI" ####
# > uniqueN( blockid2fips[,substr(blockfips,1,2)])
# [1] 52
# length(unique(EJSCREEN_Full_with_AS_CNMI_GU_VI$ST_ABBREV)) # which is in the package EJAMejscree
# [1] 56
#   dim(bgejam)
# [1] 242,940    155
#   dim(bg22)
# [1] 242,335    157
#
# so how do we get latlon for bg in as/gu/mp/vi ?  ?####

# view those block group points on a map (plot only a subset which is enough)
sam <- sample(seq_along(bgpts$bgid),5000)
plot(x = bgpts$lon[sam], y = bgpts$lat[sam], pch = '.')
```

```
# view one state, florida, where 12 are the 1st 2 digits of the FIPS:
# bgpts[bgid2fips[substr(bgfips,1,2) == '12', ], on = 'bgid']
xx='12'
mystate <- bgpts[bgid2fips[substr(bgfips, 1, 2) == xx, ], on = 'bgid'][ , .(lon, lat)]
plot(mystate, pch = '.')
rm(mystate, xx)


 How blockcounts were done:

need  data.table pkg
 bg_blockcounts <- blockwts[ , .(blockcount = uniqueN(.SD)), by=bgid]
 sum(bg_blockcounts$blockcount == 1)
   # [1] 1874 blockgroups have only 1 block
 sum(bg_blockcounts$blockcount == 1000)  the max is 1000 blocks in a bg
   # # [1] 22
 round(100*table(bg_blockcounts[blockcount <20, blockcount]) / nrow(bg_blockcounts) ,1)
   # about 1 to 3
   #   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
   # 0.8 1.2 1.3 1.4 1.5 2.1 2.2 2.4 2.6 2.8 2.8 3.0 3.0 2.9 3.0 2.9 2.8 2.7 2.5
   all.equal(bgpts$bgid, bg_blockcounts$bgid)
 bgpts[ , blockcount := bg_blockcounts$blockcount]
 dim(bgpts)
     # 242335  x    5
 usethis::use_data(bgpts) # saved for EJAM package
```

`bg_cenpop2020` *bg_cenpop2020 (DATA) data.table with all US Census 2020 block groups, Census 2020 population count, and lat/lon of Census2020-population-weighted centroid of block group*

Description

bg_cenpop2020 (DATA) data.table with all US Census 2020 block groups, Census 2020 population count, and lat/lon of Census2020-population-weighted centroid of block group

Details

also see attributes(bg_cenpop2020) for source URL and date

See Also

blockgroupstats

`blockgroupstats` *blockgroupstats (DATA) EJScreen demographic and environmental indicators for Census block groups*

Description

The EJScreen dataset (demographic, environmental indicators).

For EJ Indexes, see bgej

Details

- As of 08/2023 it was the EJScreen 2.2 version of data, which used ACS 2017-2021.

Each year this should be re-created as for the latest version. See attributes(blockgroupstats)

It is a data.table of US Census blockgroups (not blocks). With PR, and Island Areas

See https://www.epa.gov/ejscreen

Column names include bgfips, bgid (for join to blockwt$bgid), pop, pctlowinc, etc.

`censusplaces` *censusplaces (DATA) Census FIPS and other basic info on 41,414 cities/towns/places*

Description

This is just a list of US cities and similar places defined by Census

Details

from (https://www2.census.gov/geo/docs/reference/codes/PLACElist.txt) Column names: "EPA_REGION" "STATE" "ST_FIPS" "COUNTY" "CO_FIPS" "PLACE" "PL_FIPS"

`colcounter` *colcounter - Count columns with Value (at or) above (or below) threshold*

Description

colcounter - Count columns with Value (at or) above (or below) threshold

Usage

```
colcounter(
  x,
  threshold,
  or.tied = TRUE,
  na.rm = TRUE,
  below = FALSE,
  one.cut.per.col = FALSE
)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value.
threshold	numeric threshold value to compare to
or.tied	if TRUE, include ties (value in x equals threshold)
na.rm	if TRUE, used by colcounter to count only the non-NA columns in given row
below	if TRUE, count x below threshold not above threshold
one.cut.per.col	if FALSE, compare 1 threshold to all of x. If TRUE, specify one threshold per column.

Value

vector of counts as long as NROW(x)

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct() tablefixed()

Examples

```
## Not run:
 pdata <- data.frame(a=rep(80,4),b=rep(93,4), col3=c(49,98,100,100))
  ### pdata <- EJAM::blockgroupstats[ , names_e_pctile]
  ## or ## pdata <- ejscreen package file bg22[ , ejscreen package file names.e.pctile]
 pcuts <-  5 * (0:20)  # <- as.vector(keystats_e['highcut', ])
colcounter_summary(        pdata, pcuts)
colcounter_summary_pct(    pdata, pcuts)
colcounter_summary_cum(    pdata, pcuts)
colcounter_summary_cum_pct(pdata, pcuts)
colcounter_summary_cum_pct(pdata, 5 * (10:20))

x80 <- colcounter(pdata, threshold = 80, or.tied = T)
x95 <- colcounter(pdata, threshold = 95, or.tied = T)
table(x95)
tablefixed(x95, NCOL(pdata))
cbind(at80=tablefixed(x80, NCOL(pdata)), at95=tablefixed(x95, NCOL(pdata)))

## End(Not run)
```

`colcounter_summary` *colcounter_summary - Summarize how many rows have N columns at or above (or below) various thresholds? Like colcounter or cols.above.count but will handle multiple thresholds to compare to each indicator, etc. Table of counts, percents, cumulative counts, cumulative percents of places with N, or at least N, of the indicators at or above the benchmark(s)*

Description

colcounter_summary - Summarize how many rows have N columns at or above (or below) various thresholds? Like colcounter or cols.above.count but will handle multiple thresholds to compare to each indicator, etc. Table of counts, percents, cumulative counts, cumulative percents of places with N, or at least N, of the indicators at or above the benchmark(s)

Usage

```
colcounter_summary(
  x,
  thresholdlist,
  or.tied = TRUE,
  na.rm = TRUE,
  below = FALSE,
  one.cut.per.col = FALSE
)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value, like percentiles for example.
thresholdlist	vector of numeric threshold values to compare to
or.tied	if TRUE, include ties (value in x equals threshold)
na.rm	if TRUE, used by colcounter() to count only the non-NA columns in given row
below	if TRUE, count x below threshold not above threshold
one.cut.per.col	if FALSE, compare each threshold to all of x. If TRUE, specify one threshold to use for each column.

Value

A table of frequency counts

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct()

tablefixed()

Examples

```
## Not run:
 pdata <- data.frame(a=rep(80,4),b=rep(93,4), col3=c(49,98,100,100))
  ### pdata <- EJAM::blockgroupstats[ , names_e_pctile]
 pcuts <-  5 * (0:20)  # <- as.vector(keystats_e['highcut', ])
colcounter_summary(        pdata, pcuts)
colcounter_summary_pct(    pdata, pcuts)
colcounter_summary_cum(    pdata, pcuts)
colcounter_summary_cum_pct(pdata, pcuts)
colcounter_summary_cum_pct(pdata, 5 * (10:20))
a3 <- colcounter_summary_all(    pdata, pcuts)
```

```
x80 <- colcounter(pdata, threshold = 80, or.tied = T)
x95 <- colcounter(pdata, threshold = 95, or.tied = T)
table(x95)
tablefixed(x95, NCOL(pdata))
cbind(at80=tablefixed(x80, NCOL(pdata)), at95=tablefixed(x95, NCOL(pdata)))

## End(Not run)
```

`colcounter_summary_all`

colcounter_summary_all - Summarize count (and percent) of rows with exactly (and at least) N cols >= various thresholds

---

Description

Wraps 4 functions to return 4 tables: using colcounter_summary(), colcounter_summary_pct(), colcounter_summary_cum(), colcounter_summary_cum_pct()

Usage

```
colcounter_summary_all(x, thresholdlist, ...)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value, like percentiles for example.
thresholdlist	vector of numeric threshold values to compare to
...	passed to the 4 functions like or.tied=TRUE, na.rm=TRUE, below=FALSE, one.cut.per.col=FALSE

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct()

Examples

```
## Not run:
    # df <-  bg22[ , names.ej.pctile]
 df <- data.frame(a=rep(80,4),b=rep(93,4), col3=c(49,98,100,100))
 bench <- 5 * (0:20)
 a3 <- colcounter_summary_all(df, bench)
 a3[,'95',]
 a3[,,'cum_pct']
 a3['0',,]; a3[1,,]
 a3[dim(a3)[1],,]
 # a3['12',,]; a3[13,,]

 barplot(colcounter_summary_cum_pct(pdata, pcuts)[ , '80'],
    ylab='% of places', xlab='# of indicators at/above threshold',
    main='% of places with at least N/12 indicators >=80th percentile')
```

```
 barplot(colcounter_summary(pdata, pcuts)[2:13 , '95'],
    ylab='# of places', xlab='# of indicators at/above threshold',
    main='# of places with exactly N/12 indicators >=95th percentile')

 # pdata <- ejscreen package file bg22[ , names.e.pctile]
 colcounter_summary_cum_pct(pdata,c(50,80,90,95))
 xs <- 1:12
 plot(x=xs, y=colcounter_summary_cum_pct(pdata, 50)[xs+1],
  type='b', col='gray', ylim=c(0, 100),
   main='% of places with at least x/12 indicators >=Nth percentile',
    ylab='% of places', xlab='# of indicators')
 points(xs, colcounter_summary_cum_pct(pdata, 80)[xs+1], type='b', col='blue')
 points(xs, colcounter_summary_cum_pct(pdata, 90)[xs+1], type='b', col='orange')
 points(xs, colcounter_summary_cum_pct(pdata, 95)[xs+1], type='b', col='red')
 legend(x = 'topright', legend = paste0('>= ', c(50, 80, 90, 95),'th percentile'),
  fill = c('gray', 'blue', 'orange', 'red'))

 # pdata <- bg22[ ,  names.ej.pctile]
 colcounter_summary_cum_pct(pdata,c(50,80,90,95))
 xs <- 1:12
 plot(x=xs, y=colcounter_summary_cum_pct(pdata, 50)[xs+1],
   type='b', col='gray', ylim=c(0, 40),
  main='% of places with at least x/12 indicators >=Nth percentile', ylab='% of places',
   xlab='# of indicators')
 points(xs, colcounter_summary_cum_pct(pdata, 80)[xs+1], type='b', col='blue')
 points(xs, colcounter_summary_cum_pct(pdata, 90)[xs+1], type='b', col='orange')
 points(xs, colcounter_summary_cum_pct(pdata, 95)[xs+1], type='b', col='red')
 legend(x = 'topright', legend = paste0('>= ', c(50, 80, 90, 95),'th percentile'),
   fill = c('gray', 'blue', 'orange', 'red'))

## End(Not run)
```

`colcounter_summary_cum`

colcounter_summary_cum - Summarize how many rows have AT LEAST N columns at or above (or below) various thresholds See colcounter_summary() for more info and examples.

Description

colcounter_summary_cum - Summarize how many rows have AT LEAST N columns at or above (or below) various thresholds See colcounter_summary() for more info and examples.

Usage

```
colcounter_summary_cum(
  x,
  thresholdlist,
  or.tied = TRUE,
  na.rm = TRUE,
  below = FALSE,
  one.cut.per.col = FALSE
)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value, like percentiles for example.
thresholdlist	vector of numeric threshold values to compare to
or.tied	if TRUE, include ties (value in x equals threshold)
na.rm	if TRUE, used by colcounter to count only the non-NA columns in given row
below	if TRUE, count x below threshold not above threshold
one.cut.per.col	if FALSE, compare each threshold to all of x. If TRUE, specify one threshold to use for each column.

Value

A table of cumulative frequency counts

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct()

---

colcounter_summary_cum_pct

colcounter_summary_cum_pct - Summarize what percent of rows have AT LEAST N columns at or above (or below) various thresholds

---

Description

colcounter_summary_cum_pct - Summarize what percent of rows have AT LEAST N columns at or above (or below) various thresholds

Usage

```
colcounter_summary_cum_pct(x, thresholdlist, ...)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value, like percentiles for example.
thresholdlist	vector of numeric threshold values to compare to
...	passed to colcounter_summary_cum() like or.tied=TRUE, na.rm=TRUE, below=FALSE, one.cut.per.col=FALSE

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct()

colcounter_summary_pct

colcounter_summary_pct - Summarize what percent of rows have N columns at or above (or below) various thresholds

Description

colcounter_summary_pct - Summarize what percent of rows have N columns at or above (or below) various thresholds

Usage

```
colcounter_summary_pct(x, thresholdlist, ...)
```

Arguments

x	Data.frame or matrix of numbers to be compared to threshold value, like percentiles for example.
thresholdlist	vector of numeric threshold values to compare to
...	passed to colcounter_summary() like or.tied=TRUE, na.rm=TRUE, below=FALSE, one.cut.per.col=FALSE

Details

See examples for colcounter_summary_cum_pct()

See Also

colcounter_summary_all() colcounter_summary() colcounter_summary_cum() colcounter_summary_pct() colcounter_summary_cum_pct()

counties_as_sites

counties_as_sites - Analyze US Counties as if they were sites, to get EJ indicators summary for each county

Description

counties_as_sites - Analyze US Counties as if they were sites, to get EJ indicators summary for each county

Usage

```
counties_as_sites(fips)
```

Arguments

fips	County FIPS vector (ideally as character not numeric values)

Details

This function provides one row per blockgroup. getblocksnearby_from_fips() provides one row per block.

Value

data.table with one row per blockgroup in these counties, or all pairs of county fips - bgid, and a unique siteid assigned to each county

See Also

getblocksnearby_from_fips()

Examples

```
counties_as_sites(c('01001','72153'))
# Largest US Counties by ACS Population Totals:
blockgroupstats[ , .(ST = ST[1], countypop = sum(pop)),
 by=.(FIPS = substr(bgfips,1,5))][order(-countypop),][1:20, .(
 CountyPopulation = prettyNum(countypop, big.mark = ","), FIPS, ST)]
```

---

count_sites_with_n_high_scores

count_sites_with_n_high_scores

Description

count_sites_with_n_high_scores

Usage

```
count_sites_with_n_high_scores(
  scores,
  thresholds = c(1.01, 1.5, 2, 3, 5, 10),
  xwide = c("statewide", "nationwide")[1]
)
```

Arguments

scores	score
thresholds	thresholds
xwide	xwide

dataload_from_aws *dataload_from_aws - utility to load datasets from AWS DMAP Data Commons, into memory*

Description

dataload_from_aws - utility to load datasets from AWS DMAP Data Commons, into memory

Usage

```
dataload_from_aws(
 varnames = c("bgid2fips", "blockid2fips", "blockpoints", "blockwts", "quaddata"),
  ext = c(".arrow", ".rda")[2],
  fun = c("arrow::read_ipc_file", "load")[2],
  envir = globalenv(),
  mybucket = "dmap-data-commons-oa",
  mybucketfolder = "EJAM",
  folder_local_source = "~/../Downloads",
  justchecking = FALSE,
  check_server_even_if_justchecking = TRUE,
  testing = FALSE
)
```

Arguments

varnames	character vector of the quoted names of the data objects like blockwts or quaddata
envir	e.g., globalenv() or parent.frame()
mybucket	where in AWS, like
mybucketfolder	where in AWS, like EJAM
folder_local_source	path of folder (not ending in forward slash) to look in for locally saved copies during development to avoid waiting for download from a server.
justchecking	set to TRUE to get object size (and confirm file is accessible/exists)
check_server_even_if_justchecking	set this to TRUE to stop checking server to see if files are there when justchecking = TRUE. But server is always checked if justchecking = FALSE.
testing	only for testing

Details

See source code for details.

*** IF in interactive() mode, tries dataload_from_local() first during development to avoid slow downloads.

Also see https://shiny.posit.co/r/articles/improve/scoping/

Does it require credentials?

Use dataload_from_aws(justchecking=TRUE),

or datapack("EJAM") to get info,

or tables(),

or object.size(quaddata)

NOTE: blockid2fips is HUGE in memory, and is used only in state_from_blocktable() and state_from_blockid(), which are not always needed by the app, so maybe should not load this unless/until needed?

blockid2fips is roughly 600 MB in RAM because it stores 8 million block FIPS as text.

List 9/2023 was:

- blockid2fips (20 MB on disk, approx 600 MB RAM !!)
- quaddata (168 MB on disk, 218 MB RAM)
- blockpoints (86 MB on disk, 156 MB RAM)
- blockwts (31 MB on disk, 125 MB RAM)
- bgid2fips (18 MB RAM)

Value

nothing - just loads data into environment (unless justchecking=T)

See Also

`datapack()` `dataload_from_aws()` `dataload_from_package()` `indexblocks()` `.onAttach()`

`dataload_from_entirefolder` *dataload_from_entirefolder loads into global environment all .rda files found in specified folder*

Description

dataload_from_entirefolder loads into global environment all .rda files found in specified folder

Usage

```
dataload_from_entirefolder(folder = "./data")
```

Arguments

`folder`	path

Value

nothing. just loads to global envt

dataload_from_local *dataload_from_local - load datasets from local disk folder utility for analysts/developers to store large block/other data locally instead of redownloading*

Description

dataload_from_local - load datasets from local disk folder utility for analysts/developers to store large block/other data locally instead of redownloading

Usage

```
dataload_from_local(
  varnames = c(c("blockwts", "blockpoints", "blockid2fips", "quaddata"), "bgej",
    "bgid2fips", c("frs", "frs_by_programid", "frs_by_naics", "frs_by_sic",
    "frs_by_mact"))[1:4],
  ext = c(".arrow", ".rda")[1],
  fun = c("arrow::read_ipc_file", "load")[1],
  envir = globalenv(),
  folder_local_source = NULL,
  justchecking = FALSE,
  testing = FALSE
)
```

Arguments

varnames	use defaults, or vector of names like "bgej" or use "all" to get all available
ext	use defaults
fun	use defaults
envir	use defaults. see dataload_from_pins()
folder_local_source	Your local folder path. see dataload_from_pins()
justchecking	use defaults. see dataload_from_pins()
testing	use defaults

Details

See dataload_from_pins() also.

rm(bgid2fips, blockid2fips, blockpoints, blockwts, quaddata)

dataload_from_local(folder_local_source = '.')

Value

vector of paths to files (as derived from varnames) that were actually found in folder_local_source, but only for those not already in memory, so it is just the ones loaded from disk because not already in memory and found on disk locally.

dataload_from_package *dataload_from_package - utility to load a couple of datasets using data immediately instead of relying on lazy loading*

Description

dataload_from_package - utility to load a couple of datasets using data immediately instead of relying on lazy loading

Usage

```
dataload_from_package(
  olist = c("blockgroupstats", "usastats", "statestats"),
  envir = globalenv()
)
```

Arguments

olist	vector of strings giving names of objects to load using data(). This could also include other large datasets that are slow to lazyload but not always needed: "frs", "frs_by_programid ", "frs_by_naics", etc.
envir	the environment into which they should be loaded

Details

See also read_builtin() function from the readr package!

Default is to load some but not all the datasets into memory immediately. blockgroupstats, usastats, statestats, and some others are always essential to EJAM, but frs and frs_by_programid are huge datasets (and frs_by_sic and frs_by_naics) and not always used - only to find regulated facilities by ID, etc. The frs-related datasets here can be roughly 1.5 GB in RAM, perhaps.

Value

Nothing

See Also

datapack() dataload_from_aws() dataload_from_package() indexblocks() .onAttach()

Examples

```
 x <- datapack("EJAM")
 subset(x, x$size >= 0.1) # at least 100 KB
grep("names_", x$Item, value = T, ignore.case = T, invert = T) # most were like names_d, etc.
 ls()
data("avg.in.us", package="EJAM") # lazy load an object into memory and make it visible to user
 ls()
 rm(avg.in.us, x)
```

dataload_from_pins *dataload_from_pins - download / load datasets from pin board*

Description

dataload_from_pins - download / load datasets from pin board

Usage

```
dataload_from_pins(
  varnames = c(c("blockwts", "blockpoints", "blockid2fips", "quaddata"), "bgej",
    "bgid2fips", c("frs", "frs_by_programid", "frs_by_naics", "frs_by_sic",
    "frs_by_mact"))[1:4],
  boardfolder = "Mark",
  auth = "auto",
  server = "https://rstudio-connect.dmap-stage.aws.epa.gov",
  folder_local_source = NULL,
  envir = globalenv(),
  justchecking = FALSE
)
```

Arguments

varnames	character vector of names of R objects to get from board, or set this to "all" to load all of them
boardfolder	if needed to specify a different folder than default
auth	See help documentation for pins::board_connect()
server	if needed to specify a server other than default (which might be stored in envt variable CONNECT_SERVER or be registered via the rsconnect package). Note if auth = "envvar" then it looks for CONNECT_SERVER to get name of server which needs to be the full url starting with https:// - see help for board_connect
folder_local_source	path of local folder to look in for locally saved copies in case pins board is not reachable by user.
envir	if needed to specify environment other than default, e.g., globalenv() or parent.frame()
justchecking	can set to TRUE to just see a list of what pins are stored in that board

Details

This does work if on VPN and if credentials already set up for the user doing this:

board <- pins::board_connect(auth = "rsconnect")

This does work if that is true plus the two environment variables were created:

board <- pins::board_connect(auth = 'manual',

server = Sys.getenv("CONNECT_SERVER"),

key = Sys.getenv("CONNECT_API_KEY")

)

after Sys.setenv(CONNECT_SERVER = "https://rstudio-connect.dmap-stage.aws.epa.gov")

and Sys.setenv(CONNECT_API_KEY = correct-API-key-goes-here)

Value

If justchecking = FALSE,

returns vector of names of objects now in memory in specified envir, either because

1. already in memory or
2. loaded from local disk or
3. successfully downloaded.

If justchecking = TRUE, however,

returns vector of names of ALL objects found in specified pin board (or NULL if cannot connect)

regardless of whether they are already in the environment, and

regardless of whether they were specified among varnames, or are related to EJAM at all.

---

datapack *datapack - See info about the data sets in one or more packages - internal utility function Wrapper for data() and gets memory size of objects and silently returns a data.frame*

Description

datapack - See info about the data sets in one or more packages - internal utility function Wrapper for data() and gets memory size of objects and silently returns a data.frame

Usage

```
datapack(pkg = ejampackages, len = 30, sortbysize = TRUE)
```

Arguments

pkg	a character vector giving the package(s) to look in for data sets
len	Only affects what is printed to console - specifies the number of characters to limit Title to, making it easier to see in the console.
sortbysize	if TRUE, sort by increasing size of object, within each package, not alpha.

Details

do not rely on this much - it was a quick utility. it also creates and leaves in global envt objects in packages

Value

data.frame with Item and Title as columns

Examples

```
datapack("datasets")
datapack("MASS")
y = datapack("EJAM")
x = datapack(c("EJAM", "EJAMejscreenapi", "EJAMbatch.summarizer"))
x[order(x$Package, x$Item), 1:3]
tail(x[ , 1:3], 20)
```

`datawrite_to_aws` *datawrite_to_aws - NOT YET WORKING - AccessDenied Write object(s) like a dataset to DMAP Data Commons, formatted as .arrow or .rda*

Description

datawrite_to_aws - NOT YET WORKING - AccessDenied Write object(s) like a dataset to DMAP Data Commons, formatted as .arrow or .rda

Usage

```
datawrite_to_aws(
 varnames = c("bgid2fips", "blockid2fips", "blockpoints", "blockwts", "quaddata"),
  ext = c(".arrow", ".rda")[2],
  fun = c("arrow::write_ipc_file", "save")[2],
  mybucket = "dmap-data-commons-oa",
  mybucketfolder = "EJAM",
  justchecking = TRUE
)
```

Arguments

`varnames`	vector of object names to upload
`ext`	file .extension appropriate to the format and fun, like ".rda" or ".arrow"
`fun`	function to use, but as a character string, like "arrow::write_ipc_file" but fun is ignored if ext=".rda" since it then just uses s3save()
`mybucket`	do not need to change
`mybucketfolder`	do not need to change
`justchecking`	set this to FALSE to actually upload instead of just viewing in console the commands to be used, to test/check this

Details

mybucket <- 'dmap-data-commons-oa' #

bucket_contents <- data.table::rbindlist(

`get_bucket(bucket = mybucket, prefix = "EJAM"),`

`fill = TRUE`

)

bucket_contents

Value

the paths of the objects on server

`datawrite_to_local` *datawrite_to_local Write large object(s) like EJAM datasets to local disk for convenience during app/pkg development, formatted as .arrow or .rda*

Description

datawrite_to_local Write large object(s) like EJAM datasets to local disk for convenience during app/pkg development, formatted as .arrow or .rda

Usage

```
datawrite_to_local(
  varnames = c("bgid2fips", "blockid2fips", "blockpoints", "blockwts", "quaddata"),
   ext = c(".arrow", ".rda")[1],
   folder_local_source = "~/../Downloads",
   fun = c("arrow::write_ipc_file", "save")[1],
   justchecking = F,
   overwrite = FALSE
)
```

Arguments

`varnames`	vector of object names
`ext`	file .extension appropriate to the format, ".rda" or ".arrow"
`folder_local_source`	path to local folder without slash at end
`fun`	function to use, but as a character string, like "arrow::write_ipc_file" but fun is ignored if ext=".rda" since it then just uses save()
`justchecking`	set this to FALSE to actually save instead of just seeing in console info or the commands to be used, to test/check this
`overwrite`	Set to TRUE to overwrite file if it exists already, with new copy.

Value

the paths of the objects as requested to be saved whether or not actually done

See Also

`datawrite_to_aws()` `datawrite_to_local()` `dataload_from_local()` `dataload_from_aws()`

Examples

```
# datawrite_to_local(ext = ".arrow", folder_local_source = ".", justchecking = F, overwrite = T)
```

distance_by_group *distance_by_group - Avg distance of each demog group (of multiple groups) Same as* plot_distance_mean_by_group() *but no plot by default*

Description

distance_by_group - Avg distance of each demog group (of multiple groups) Same as plot_distance_mean_by_group() but no plot by default

Usage

```
distance_by_group(
  results_bybg_people,
  demogvarname = NULL,
  demoglabel = NULL,
  returnwhat = "table",
  graph = FALSE
)
```

Arguments

results_bybg_people	data.table from doaggregate()$results_bybg_people
demogvarname	vector of column names like "pctlowinc" etc.
demoglabel	vector of labels like "Low Income Residents" etc.
returnwhat	If returnwhat is "table", invisibly returns a data.frame with group, ratio, avg_distance_for_group, avg_distance_for_nongroup. If returnwhat is "plotfilename" then it returns the full path including filename of a .png in a tempdir If returnwhat is "plot" then it returns the plot object as needed for table_xls_format() ?

Details

see examples in plot_distance_cdf_by_group()

Value

see parameter returnwhat

See Also

distance_by_group()

distance_by_group_plot() plot_distance_cdf_by_group()

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
```

```
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

`distance_by_group1` *distance_by_group1 - JUST ONE GROUP Get average distance for ONE demographic group versus everyone else*

Description

distance_by_group1 - JUST ONE GROUP Get average distance for ONE demographic group versus everyone else

Usage

```
distance_by_group1(
  results_bybg_people,
  demogvarname = "Demog.Index",
  demoglabel = demogvarname
)
```

Arguments

`results_bybg_people`	data.table from doaggregate()$results_bybg_people
`demogvarname`	e.g., "pctlowinc"
`demoglabel`	e.g., "Low Income Residents"

Details

Note on Avg Distance and range of distances in each Demog group, & %D as function of distance:

We have info on each blockgroup near each site, which means some small % of those bgs are duplicated in this table:

`results_bybg_people`

Mostly we want overall (not by site) to know avg and cum distrib of distances in each demog,

(and also %D as a function of continuous distance),

and for those stats we would want to take only unique blockgroups from here, using the shorter distance, so the distribution of distances does not doublecount people.

But we might also want to see that distribution of distances by D for just 1 site?

And we might also want to see the %D as a function of continuous distance at just 1 site?

So to retain flexibility doaggregate() reports all instances of blockgroup-site pairings.

Value

list of 2 numbers: avg_distance_for_group and avg_distance_for_nongroup

See Also

plot_distance_mean_by_group() distance_by_group()

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

---

distance_by_group_by_site

distance_by_group_by_site - DRAFT FUNCTION

Description

distance_by_group_by_site - DRAFT FUNCTION

Usage

```
distance_by_group_by_site(bybg)
```

Arguments

bybg	such as ejamit()$results_bybg_people

Value

table of ratios, one col per site, one row per indicator

See Also

plot_distance_cdf_by_group() plot_distance_mean_by_group() distance_by_group() distance_mean_by_gr

distance_by_group_plot

distance_by_group_plot or plot_distance_cdf_by_group

Description

distance_by_group_plot or plot_distance_cdf_by_group

Usage

```
distance_by_group_plot(...)
```

Value

see returnwhat parameter

See Also

`distance_by_group()` `ejamit()` for examples

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

distance_cdf_by_group_plot

distance_cdf_by_group_plot - SLOW - needs to be optimized Plot a graphic showing cumulative shares of ONLY ONE demographic group that are within each distance

Description

distance_cdf_by_group_plot - SLOW - needs to be optimized Plot a graphic showing cumulative shares of ONLY ONE demographic group that are within each distance

Usage

```
distance_cdf_by_group_plot(
  results_bybg_people,
 radius_miles = round(max(results_bybg_people$distance_min_avgperson, na.rm = T), 1),
  demogvarname = "Demog.Index",
  demoglabel = demogvarname,
  color1 = "red",
  color2 = "black"
)
```

Arguments

results_bybg_people	data.table from doaggregate()$results_bybg_people
radius_miles	miles radius that was max distance analyzed
demogvarname	name of column in results_bybg_people, e.g., "pctlowinc"
demoglabel	friendly text name for labelling graphic, like "Low income residents"
color1	color like "red" for demographic group of interest
color2	color like "gray" for everyone else

Value

invisibly returns full table of sorted distances of blockgroups, cumulative count of demog group at that block group's distance, and cumulative count of everyone else in that block group

See Also

distance_by_group() getblocksnearbyviaQuadTree() for examples

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

distance_mean_by_group

distance_mean_by_group - Avg distance of each demog group (of multiple groups) Same as `plot_distance_mean_by_group()` *but no plot by default*

Description

distance_mean_by_group - Avg distance of each demog group (of multiple groups) Same as plot_distance_mean_by_g but no plot by default

Usage

```
distance_mean_by_group(
  results_bybg_people,
  demogvarname = NULL,
  demoglabel = NULL,
  returnwhat = "table",
  graph = FALSE
)
```

Arguments

results_bybg_people	data.table from doaggregate()$results_bybg_people
demogvarname	vector of column names like "pctlowinc" etc.
demoglabel	vector of labels like "Low Income Residents" etc.
returnwhat	If returnwhat is "table", invisibly returns a data.frame with group, ratio, avg_distance_for_group, avg_distance_for_nongroup. If returnwhat is "plotfilename" then it returns the full path including filename of a .png in a tempdir If returnwhat is "plot" then it returns the plot object as needed for table_xls_format() ?

Details

Note that the ratio shown is a ratio of distance among others to distance of a given group, so values below 1 mean the given demographic group lives closer to facilities. A value of 0.85 would mean the group is only 85% as far from a site as everyone else.

Note it is in miles assuming input was in miles, and the distance for each resident is actually the average distance of all residents within their Census block (not block group), and when a site is very close to the block internal point (like a centroid) relative to the size of the block, the distance to the average resident in the block is estimated as 90 percent of the effective radius, which is what the radius of the block would be if it were the same area in square meters or miles but circular in shape. This is the approach used in EJScreen to estimate average proximity of a block resident in cases where the block is extremely close to the site or the site may actually be inside the block, or exactly on top of the internal point of the block, in which case zero would not be an appropriate estimate of the distance, hence this adjustment is made in EJAM getblocksnearby()

Value

see parameter returnwhat

See Also

distance_by_group()

distance_by_group_plot() plot_distance_cdf_by_group()

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

---

distance_via_surfacedistance

distance_via_surfacedistance Convert surface distance to actual distance

---

Description

\preformatted{
    Just a simple formula:
   earthRadius_miles <- 3959
   angle_rad <- x/earthRadius_miles
   # Calculate radius * cord length
   return(earthRadius_miles * 2*sin(angle_rad/2))
   }

Usage

```
distance_via_surfacedistance(x)
```

Arguments

x	surface distance in miles

doaggregate *doaggregate - Summarize indicators in each buffer (given the blocks in each buffer and indicators for each block)*

Description

This updated 2023 code takes a set of facilities and the set of blocks that are near each, (as identified previously, in other code that has identified which blocks are nearby) and combines those with indicator scores for block groups.

Usage

```
doaggregate(
  sites2blocks,
  sites2states_or_latlon = NA,
  radius = NULL,
  countcols = NULL,
  popmeancols = NULL,
  calculatedcols = NULL,
  subgroups_type = "nh",
  include_ejindexes = FALSE,
  calculate_ratios = TRUE,
  extra_demog = TRUE,
  need_proximityscore = FALSE,
  infer_sitepoints = FALSE,
  called_by_ejamit = FALSE,
  updateProgress = NULL,
  silentinteractive = TRUE,
  testing = FALSE,
  ...
)
```

Arguments

sites2blocks	data.table of distances in miles between all sites (facilities) and nearby Census block internal points, with columns siteid, blockid, distance, created by getblocksnearby function. See sites2blocks_example10pts_1miles aka testoutput_getblocksnearby_10pts_1miles dataset in package, as input to this function
sites2states_or_latlon	data.table or just data.frame, with columns siteid (each unique one in sites2blocks) and ST (2-character State abbreviation) or lat and lon
radius	Optional radius in miles to limit analysis to. By default this function uses all the distances that were provided in the output of getblocksnearby(), and reports radius estimated as rounded max of distance values in inputs to doaggregate. But there may be cases where you want to run getblocksnearby() once for 10 miles, say, on a very long list of sites (1,000 or more, say), and then get summary results for 1, 3, 5, and 10 miles without having to redo the getblocksnearby() part for each radius. This lets you just run getblocksnearby() once for the largest radius, and then query those results to get doaggregate() to summarize at any distance that is less than or equal to the original radius analyzed by getblocksnearby().

countcols — character vector of names of variables to aggregate within a buffer using a sum of counts, like, for example, the number of people for whom a poverty ratio is known, the count of which is the exact denominator needed to correctly calculate percent low income.

popmeancols — character vector of names of variables to aggregate within a buffer using population weighted mean.

calculatedcols — character vector of names of variables to aggregate within a buffer using formulas that have to be specified.

subgroups_type — Optional (uses default). Set this to "nh" for non-hispanic race subgroups as in Non-Hispanic White Alone, nhwa and others in names_d_subgroups_nh; "alone" for EJScreen v2.2 style race subgroups as in White Alone, wa and others in names_d_subgroups_alone; "both" for both versions. Possibly another option is "original" or "default" but work in progress.

include_ejindexes — whether to calculate EJ Indexes and return that information

calculate_ratios — whether to calculate and return ratio of each indicator to its US and State overall mean

extra_demog — if should include more indicators from EJScreen v2.2 report, on language, more age groups, gender, percent with disability, poverty, etc.

need_proximityscore — whether to calculate proximity scores

infer_sitepoints — set to TRUE to try to infer the lat,lon of each site around which the blocks in sites2blocks were found. lat,lon of each site will be approximated as average of nearby blocks, although a more accurate slower way would be to use reported distance of each of 3 of the furthest block points and triangulate

called_by_ejamit — Set to TRUE by ejamit() to suppress some outputs even if ejamit(silentinteractive=F)

updateProgress — progress bar function used for shiny app

silentinteractive — Set to TRUE to see results in RStudio console. Set to FALSE to prevent long output showing in console in RStudio when in interactive mode

testing — used while testing this function

... — more to pass to another function? Not used currently.

Details

For all examples, see `getblocksnearbyviaQuadTree()`

This function aggregates the blockgroup scores to create a summary of each indicator, as a raw score and US percentile and State percentile, in each buffer (i.e., near each facility):

- **SUMS OF COUNTS**: for population count, or number of households or Hispanics, etc.
- **POPULATION-WEIGHTED MEANS**: for Environmental indicators.
  EJ Indexes*:* The way EJScreen does this is apparently finding the pop wtd mean of EJ Index raw scores, not the EJ Index formula applied to the summarized demographic score and aggregated envt number.
- **CALCULATED BY FORMULA**: Buffer or overall score calculated via formulas using aggregated counts, such as percent low income = sum of counts low income / sum of counts of denominator, which in this case is the count of those for whom the poverty ratio is known.

- **LOOKED UP**: Aggregated scores are converted into percentile terms via lookup tables (US or State version).

This function requires the following datasets:

- blockwts: data.table with these columns: blockid , bgid, blockwt
- quaddata data.table used to create localtree, a quad tree index of block points (and localtree that is created when package is loaded)
- blockgroupstats - A data.table (such as EJScreen demographic and environmental data by blockgroup?)

Value

list with named elements:

- `results_overall` one row data.table, like results_by_site, but just one row with aggregated results for all unique residents.
- `results_by_site` results for individual sites (buffers) - a data.table of results, one row per siteid, one column per indicator
- **results_bybg_people** results for each block group, to allow for showing the distribution of each indicator across everyone within each demographic group.
- **longnames** descriptive long names for the indicators in the above outputs
- **count_of_blocks_near_multiple_sites** additional detail

See Also

ejamit `getblocksnearby()`

---

dupenames *dupenames - UTILITY - check conflicting exported function or data names*

Description

See what same-named objects (functions or data) are exported by some (installed) packages

Usage

```
dupenames(
  pkg = EJAM::ejampackages,
  sortbypkg = FALSE,
  compare.functions = TRUE
)
```

Arguments

`pkg`	one or more package names as vector of strings. If "all" it checks all installed pkgs, but takes very very long potentially.
`sortbypkg`	If TRUE, just returns same thing but sorted by package name
`compare.functions`	If TRUE, sends to console inf about whether body and formals of the functions are identical between functions of same name from different packages. Only checks the first 2 copies, not any additional ones (where 3+ pkgs use same name)

Details

utility to find same-named exported objects (functions or datasets) within source code of 2+ packages, and see what is on search path, for dev renaming / moving functions/ packages

Value

data.frame with columns Package, Object name (or NA if no dupes)

See Also

`all_equal_functions()`

---

dupeRfiles *dupeRfiles - UTILITY - check conflicting sourcefile names*

Description

See what same-named .R files are in 2 sourcecode folders

Usage

```
dupeRfiles(folder1 = "../EJAM/R", folder2 = "./R")
```

Arguments

`folder1`	path to other folder with R source files
`folder2`	path to a folder with R source files, defaults to "./R"

Details

useful for shiny app that is not a package, as ejamlite and EJAMejscreenapi had copies of some EJAM files.

See `dupenames()` for when they are all packages.

---

ejam2excel *ejam2excel - alias for table_xls_from_ejam()*

Description

ejam2excel - alias for table_xls_from_ejam()

Usage

```
ejam2excel(
  ejamitout,
  fname = NULL,
  save_now = TRUE,
  overwrite = TRUE,
  launchexcel = FALSE,
  interactive_console = TRUE,
  ok2plot = TRUE,
  in.testing = FALSE,
  in.analysis_title = "EJAM analysis",
  react.v1_summary_plot = NULL,
  radius_or_buffer_in_miles = NULL,
  buffer_desc = "Selected Locations",
  radius_or_buffer_description =
   "Miles radius of circular buffer (or distance used if buffering around polygons)",
  hyperlink_colnames = c("EJScreen Report", "EJScreen Map", "ECHO report"),
  ...
)
```

Arguments

ejamitout	output of ejamit()
fname	optional name or full path and name of file to save locally, like "out.xlsx"
save_now	optional logical, whether to save as a .xlsx file locally or just return workbook object that can later be written to .xlsx file using openxlsx::saveWorkbook()
overwrite	optional logical, passed to openxlsx::saveWorkbook()
launchexcel	optional logical, passed to table_xls_format(), whether to launch browser to see spreadsheet immediately
interactive_console	optional - should set to FALSE when used in code or server. If TRUE, prompts RStudio user interactively asking where to save the downloaded file
ok2plot	optional logical, passed to table_xls_format(), whether safe to try and plot or set FALSE if debugging plot problems
in.testing	optional logical
in.analysis_title	optional title as character string
react.v1_summary_plot	optional - a plot object
radius_or_buffer_in_miles	optional radius in miles
radius_or_buffer_description	optional text phrase describing places analyzed
hyperlink_colnames	optional names of columns with URLs
...	optional additional parameters passed to table_xls_format(), such as heatmap_colnames, heatmap_cuts, heatmap_colors, etc.

`ejamit` *ejamit - Get complete EJ analysis (demographic and environmental indicators) near a list of locations*

Description

This is the main function in EJAM for users who want to use EJAM from RStudio. It does essentially what the webapp does to analyze/summarize near a set of points. See help("EJAM")

Usage

```
ejamit(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = FALSE,
  quadtree = NULL,
  fips = NULL,
  shapefile_folder = NULL,
  countcols = NULL,
  popmeancols = NULL,
  calculatedcols = NULL,
  subgroups_type = "nh",
  include_ejindexes = FALSE,
  calculate_ratios = TRUE,
  extra_demog = TRUE,
  need_proximityscore = FALSE,
  infer_sitepoints = FALSE,
  need_blockwt = TRUE,
  threshold1 = 90,
  updateProgress = NULL,
  in_shiny = FALSE,
  quiet = TRUE,
  parallel = FALSE,
  silentinteractive = FALSE,
  called_by_ejamit = TRUE,
  testing = FALSE
)
```

Arguments

`sitepoints`	data.table with columns siteid, lat, lon giving point locations of sites or facilities around which are circular buffers
`radius`	in miles, defining circular buffer around a site point
`maxradius`	miles distance (max distance to check if not even 1 block point is within radius)
`avoidorphans`	logical If TRUE, then where not even 1 BLOCK internal point is within radius of a SITE, it keeps looking past radius, up to maxradius, to find nearest 1 BLOCK. What EJScreen does in that case is report NA, right? So, does EJAM really need to report stats on residents presumed to be within radius, if no block centroid is within radius? Best estimate might be to report indicators from nearest block

centroid which is probably almost always the one your site is sitting inside of, but ideally would adjust total count to be a fraction of blockwt based on what is area of circular buffer as fraction of area of block it is apparently inside of. Setting this to TRUE can produce unexpected results, which will not match EJScreen numbers. Note that if creating a proximity score, by contrast, you instead want to find nearest 1 SITE if none within radius of this BLOCK.

`quadtree` (a pointer to the large quadtree object) created using indexblocks() which uses the SearchTree package. Takes about 2-5 seconds to create this each time it is needed. It can be automatically created when the package is attached via the .onAttach() function

`fips` optional FIPS code vector to provide if using FIPS instead of sitepoints to specify places to analyze, such as a list of US Counties or tracts. Passed to `getblocksnearby_from_fips()`

`shapefile_folder`
optional path to folder that has shapefiles to analyze polygons

`countcols` character vector of names of variables to aggregate within a buffer using a sum of counts, like, for example, the number of people for whom a poverty ratio is known, the count of which is the exact denominator needed to correctly calculate percent low income.

`popmeancols` character vector of names of variables to aggregate within a buffer using population weighted mean.

`calculatedcols` character vector of names of variables to aggregate within a buffer using formulas that have to be specified.

`subgroups_type` Optional (uses default). Set this to "nh" for non-hispanic race subgroups as in Non-Hispanic White Alone, nhwa and others in names_d_subgroups_nh; "alone" for EJScreen v2.2 style race subgroups as in White Alone, wa and others in names_d_subgroups_alone; "both" for both versions. Possibly another option is "original" or "default"

`include_ejindexes`
whether to try to include EJ Indexes (assuming dataset is available) - passed to `doaggregate()`

`calculate_ratios`
whether to calculate and return ratio of each indicator to US and State overall averages - passed to `doaggregate()`

`extra_demog` if should include more indicators from v2.2 report on language etc.

`need_proximityscore`
whether to calculate proximity scores

`infer_sitepoints`
set to TRUE to try to infer the lat,lon of each site around which the blocks in sites2blocks were found. lat,lon of each site will be approximated as average of nearby blocks, although a more accurate slower way would be to use reported distance of each of 3 of the furthest block points and triangulate

`need_blockwt` if fips parameter is used, passed to `getblocksnearby_from_fips()`

`threshold1` percentile like 80 or 90 or 95 to compare percentiles to "alone" for groups like white alone (whether or not hispanic), "both" may try to include both, or possibly "original" or "default" might be added as options - passed to batch.summarize()

`updateProgress` progress bar function used for shiny app

`in_shiny` if fips parameter is used, passed to `getblocksnearby_from_fips()`

quiet	Optional. set to TRUE to avoid message about using getblock_diagnostics(), which is relevant only if a user saved the output of this function.
parallel	whether to use parallel processing in getblocksnearby() but may not be implemented yet.
silentinteractive	to prevent long output showing in console in RStudio when in interactive mode, passed to doaggregate() also. app server sets this to TRUE when calling doaggregate() but ejamit() default is to set this to FALSE when calling doaggregate().
called_by_ejamit	Set to TRUE by ejamit() to suppress some outputs even if ejamit(silentinteractive=F)
testing	used while testing this function

Value

A list of tables of results

See Also

getblocksnearby() doaggregate()

Examples

```
## Not run:
 # All in one step, using functions not shiny app:
 out <- ejamit(testpoints_100_dt, 2, quadtree=localtree)

 # Do not specify sitepoints and it will prompt you for a file,
 # if in RStudio in interactive mode!
 out <- ejamit(radius = 3)

  # Specify facilities or sites as points for test data,
  # use 1000 test facility points from the R package
  testsites <- testpoints_1000_dt
  # use facility points in an excel or csv file
  testsites <- latlon_from_anything(
   "./inst/testdata/testpoints_207_sites_with_signif_violations_NAICS_326_ECHO.csv")
  # use facility points from a random sample of EPA-regulated facilities
  testsites <- testpoints_n(1e3)

  # Specify max distance from sites to look at (residents within X miles of site point)
  radius <- 3.1 # miles

  # Get summaries of all indicators near a set of points
  out <- ejamit(testsites, radius)
  # out <- ejamit("myfile.xlsx", 3.1)

  # out2 <- ejscreenit(testpoints_05)

  # View results overall
  round(t(out$results_overall), 3.1)

  # View plots
  # plot_distance_avg_by_group(out)
  # plot_distance_cdf_by_group(out)
```

```
  # View maps
  mapfast(out$results_bysite, radius = 3.1)

  # view results at a single site
  t(out$results_bysite[1, ])
  t(out$results_bysite[out$results_bysite$siteid == 2, ])


  # if doing just 1st step of ejamit()
  #  get distance between each site and every nearby Census block
  s2b <- testdata_sites2blocks
  s2b <- getblocksnearby(testsites, radius = radius)
  s2b <- getblocksnearbyviaQuadTree(testsites, radius = radius)
  getblocks_diagnostics(s2b)
  plotblocksnearby(s2b)

  # if doing just 2d step of ejamit()
  #  get summaries of all indicators based on table of distances
  out <- doaggregate(s2b, testsites) # this works now and is simpler


## End(Not run)
```

`ejampackages` *ejampackages (DATA) list of names of key EJAM-related R packages*

Description

ejampackages (DATA) list of names of key EJAM-related R packages

---

`ejscreenapi2ejam_format` *Convert output of ejscreenapi_plus to format of ejamit table of sites Makes it easier to compare EJScreen and EJAM results*

Description

Convert output of ejscreenapi_plus to format of ejamit table of sites Makes it easier to compare EJScreen and EJAM results

Usage

```
ejscreenapi2ejam_format(
  ejscreenapi_plus_out,
  fillmissingcolumns = FALSE,
  ejamcolnames = NULL
)
```

Arguments

ejscreenapi_plus_out	results of ejscreenapi_plus() or also could be results of ejscreenit()$table even though the colnames differ, because they get converted here in that case.
fillmissingcolumns	optional. set to TRUE if you want the output to have exactly all the same columns as the EJAM table would, and fill in with NA values all the columns not provided by EJScreen API.
ejamcolnames	optional. if specified as vector of colnames, it overrides the assumed colnames that would have been taken to be colnames(testoutput_ejamit_10pts_1miles$results_bysite). Any colnames you specify here will be the colnames of the output if fillmissingcolumns = TRUE, or else those not in names(ejscreenapi_plus_out) will be omitted.

Value

A data.table not just data.frame, with some or all of the columns found in output of ejamit()$results_bysite

See Also

ejscreenapi_vs_ejam1()

Examples

```
## Not run:
   x <- ejscreenapi_plus(testpoints_10[1:2, ], radius = 1)
   y <- ejscreenapi2ejam_format(x)
   ejamvars <- names(testoutput_ejamit_10pts_1miles$results_bysite)
   all.equal(
     names(y),
     ejamvars[ejamvars %in% names(y)]
  )

  z <- ejscreenapi2ejam_format(x, fillmissingcolumns = T)
  all.equal(names(z), ejamvars)

 y2 <- ejamit(testpoints_10[1:2, ], radius = 1)
 y2 <- y2$results_bysite
 ejamvars <- names(y2)
 all.equal(names(z), ejamvars)
 sitenum <- 1
 z <- data.frame(EJAM = t(y2[sitenum, -1:-3]), EJSCREEN = t(z[sitenum, -1:-3 ]))
 z
 z[!is.na(z$EJSCREEN), ]

## End(Not run)
```

`ejscreenapi_vs_ejam1` *compare EJScreen API vs EJAM stats near one site*

Description

compare EJScreen API vs EJAM stats near one site

Usage

```
ejscreenapi_vs_ejam1(latlon, radius = 3, nadrop = FALSE, x100fix = TRUE, ...)
```

Arguments

`latlon`	1-row data.table or data.frame with colnames lat and lon
`radius`	in miles, used in ejamit() and ejscreenapi_plus()
`nadrop`	whether to drop indicators for which EJScreen API returns NA
`x100fix`	whether to multiply x100 the names_d and names_d_subgroups indicator scores to convert fractions 0 to 1 into percentages of 0 to 100, prior to rounding and reporting EJAM results here.
`...`	passed to ejamit() as any additional parameters, like include_ejindexes = FALSE

Value

a data.frame with columns EJSCREEN and EJAM, rownames are indicators like pop, blockcount_near_site, etc.

See Also

`ejscreenapi_vs_ejam1_alreadyrun()`

Examples

```
## Not run:
  pts <- testpoints_100[1:5, ]
 #z <- ejscreenapi_vs_ejam1(testpoints_100[27, ], radius = 3, nadrop = T, include_ejindexes = TRUE)
  z <- ejscreenapi_vs_ejam1(pts[5, ], radius = 3, include_ejindexes = TRUE)

  # Reported key indicators - which ones do or don't match
  # when comparing EJSCREEN and EJAM results?
  keyreportnames <- c('pop', names_these, names_pctile, names_state_pctile)
  z[z$rname %in% keyreportnames &  z$same_shown, ]
  z[z$rname %in% keyreportnames & !z$same_shown & !is.na(z$EJSCREEN), ]

  # Reported (rounded) numbers match:
  z[z$same_shown , -1]
  # Reports disagree:
  #  (and not just because of percentages being of 100 vs of 1.00)
  z[!z$same_shown & !is.na(z$EJSCREEN) & z$ratio != 0.01, -1]
  # Reports disagree if percentages reported as 0-100 vs fractions 0-1.00
  z[z$ratio == 0.01 & !is.na(z$ratio), -1]


## End(Not run)
```

`ejscreenapi_vs_ejam1_alreadyrun` *compare EJScreen API vs EJAM stats near one site (if results already run)*

Description

compare EJScreen API vs EJAM stats near one site (if results already run)

Usage

```
ejscreenapi_vs_ejam1_alreadyrun(
  apisite,
  ejamsite,
  x100fix = TRUE,
  nadrop = FALSE
)
```

Arguments

`apisite`	1-row table output of ejscreenit()$table, or ejscreenapi_plus()
`ejamsite`	1-row table output of ejamit()$results_bysite
`x100fix`	whether to multiply x100 the names_d and names_d_subgroups indicator scores to convert fractions 0 to 1 into percentages of 0 to 100, prior to rounding and reporting EJAM results here.
`nadrop`	whether to drop indicators for which EJScreen API returns NA

Value

a data.frame with columns EJSCREEN and EJAM, rownames are indicators like pop, blockcount_near_site, etc.

See Also

`ejscreenapi_vs_ejam1()`

Examples

```
## Not run:

# requires data.table
  pts <- testpoints_100[1:5, ]
  api1 <- ejscreenit(pts, radius = 3)
  api1 <- api1$table[5, ]
  ejam1 <- ejamit(pts, radius = 3, include_ejindexes = TRUE)
  ejam1 <- ejam1$results_bysite[5, ]
  z <- ejscreenapi_vs_ejam1_alreadyrun(api1, ejam1)

  # Reported key indicators - which ones do or don't match
  # when comparing EJSCREEN and EJAM results?
  keyreportnames <- c('pop', names_these, names_pctile, names_state_pctile)
  z[z$rname %in% keyreportnames &  z$same_shown, ]
```

```
  z[z$rname %in% keyreportnames & !z$same_shown & !is.na(z$EJSCREEN), ]

  # Reported (rounded) numbers match:
  z[z$same_shown , -1]
  # Reports disagree:
  #  (and not just because of percentages being of 100 vs of 1.00)
  z[!z$same_shown & !is.na(z$EJSCREEN) & z$ratio != 0.01, -1]
  # Reports disagree if percentages reported as 0-100 vs fractions 0-1.00
  z[z$ratio == 0.01 & !is.na(z$ratio), -1]

## End(Not run)
```

`ejscreenit_for_ejam` *ejscreenit_for_ejam Wrapper for ejscreenit() from EJAMejscreenapi package, to use in EJAM app*

Description

ejscreenit_for_ejam Wrapper for ejscreenit() from EJAMejscreenapi package, to use in EJAM app

Usage

```
ejscreenit_for_ejam(sitepoints, radius = 3, fillmissingcolumns = TRUE, ...)
```

Arguments

`sitepoints`	table with lat and lon columns
`radius`	in miles
`...`	passed to ejscreenit()

Value

a data.table that looks like output of ejamit()$results_bysite

See Also

`ejscreenapi_vs_ejam1()`

`fips2countyname` *fips2countyname - Get county names from county FIPS codes Get county names from county FIPS codes*

Description

fips2countyname - Get county names from county FIPS codes Get county names from county FIPS codes

Usage

```
fips2countyname(fips, includestate = c("ST", "Statename", "")[1])
```

Arguments

fips	vector of US Census FIPS codes for Counties (5 digits each). can be string or numeric, with or without leading zeroes.
includestate	can be ST, Statename, "", or TRUE to specify what if anything comes after county name and comma

Value

vector of county names, optionally with comma and 2-character abbreviation or full state name.

Examples

```
# names of all counties in ME and NY
fips2countyname(fips_counties_from_state_abbrev(c("ME", "NY")), includestate = "ST")
 fips_counties_from_state_abbrev(c("AK", "LA"))
```

---

fips2name — *fips2name - Get county or state names from county or state FIPS codes Get county or state names from county or state FIPS codes*

Description

fips2name - Get county or state names from county or state FIPS codes Get county or state names from county or state FIPS codes

Usage

```
fips2name(fips, ...)
```

Arguments

fips	vector of US Census FIPS codes for Counties (5 digits each) or States (2 digits). Can be string or numeric, with or without leading zeroes.
...	passed to fips2countyname() to control whether it appends something like , NY or , New York after county name

Value

vector of state and/or county names, where county names optionally have comma and 2-character abbreviation or full state name.

Examples

```
 fips2name(fips_counties_from_state_abbrev(c("AK", "LA"))  )
fips2name(c(22, 02013)) # can have mix where some are a whole state and others are a county.
```

`fips2statename` *fips2statename - Get state names from any type of FIPS codes Get state names from any type of FIPS codes*

Description

fips2statename - Get state names from any type of FIPS codes Get state names from any type of FIPS codes

Usage

```
fips2statename(fips)
```

Arguments

`fips`	vector of FIPS

Value

vector of state names

Examples

```
cbind(fips_lead_zero(1:80), fips2state_abbrev(1:80), fips2statename(1:80))
```

`fips2state_abbrev` *fips2state_abbrev - Get state abbreviations from any type of FIPS codes Get state abbreviations from any type of FIPS codes*

Description

fips2state_abbrev - Get state abbreviations from any type of FIPS codes Get state abbreviations from any type of FIPS codes

Usage

```
fips2state_abbrev(fips)
```

Arguments

`fips`	vector of FIPS

Value

vector of abbreviations like "NY", "LA", "DE", etc.

Examples

```
cbind(
  stfips    = fips_lead_zero(1:80),
  ST     = fips2state_abbrev(1:80),
  statename = fips2statename(1:80)
)
```

`fips2state_fips` *fips2state_fips - Get state FIPS codes from any type of FIPS codes Get state FIPS codes from any type of FIPS codes*

Description

fips2state_fips - Get state FIPS codes from any type of FIPS codes Get state FIPS codes from any type of FIPS codes

Usage

```
fips2state_fips(fips)
```

Arguments

`fips`	vector of FIPS

Details

Tells you which State contains each County (or tract or blockgroup or block)

Value

vector of State FIPS 2 characters each

Examples

```
fips2state_fips(fips_counties_from_statename(c("Delaware", "Rhode Island")))
```

`fipstype` *fipstype - identify what type of Census geography is each FIPS code (block, county, etc.)*

Description

fipstype - identify what type of Census geography is each FIPS code (block, county, etc.)

Usage

```
fipstype(fips)
```

Arguments

`fips`	vector of one or more Census FIPS with or without leading zeroes, as strings or numeric

Value

vector of types: "block", "blockgroup", "tract", "county", or "state"

Examples

```
fips_counties_from_statename(c("Connecticut", "Delaware") )
# [1] "09001" "09003" "09005" "09007" "09009" "09011" "09013" "09015" "10001" "10003" "10005"
fipstype(9001)
fipstype("10001")
# note blockid2fips is a large file, but can be obtained via [dataload_from_pins()]
## Not run:
fipsexamples <- c(
  fips_state_from_statename("Alaska"),
  fips_counties_from_state_abbrev("DE")[1],
   substr(blockid2fips$blockfips[1],1,11),
   blockgroupstats$bgfips[1],
   blockid2fips$blockfips[1]
)
cbind(fipsexamples, type = fipstype(fipsexamples))

## End(Not run)
```

`fips_bg_from_anyfips` *fips_bg_from_anyfips - Get unique blockgroup fips in or containing specified fips of any type Convert any FIPS codes to the FIPS of all the blockgroups that are among or within or containing those FIPS*

Description

fips_bg_from_anyfips - Get unique blockgroup fips in or containing specified fips of any type Convert any FIPS codes to the FIPS of all the blockgroups that are among or within or containing those FIPS

Usage

```
fips_bg_from_anyfips(fips)
```

Arguments

`fips`	vector of US FIPS codes, as character or numeric, with or without their leading zeroes, each with as many characters

Details

This is a way to get a list of blockgroups, specified by state/county/tract or even block.

Takes a vector of one or more FIPS that could be State (2-digit), County (5-digit), Tract (11-digit), or blockgroup (12 digit), or even block (15-digit fips).

Returns unique vector of FIPS of all US blockgroups (including DC and Puerto Rico) that contain any specified blocks, are equal to any specified blockgroup fips, or are contained within any provided tract/county/state FIPS.

Value

vector of blockgroup FIPS (or NA values) that may be much longer than the vector of fips passed to this function.

See Also

fips_lead_zero()

Examples

```
# all blockgroups in one state
fips_counties_from_state_abbrev("DE")
fips_bg_from_anyfips( fips_counties_from_state_abbrev("DE") )

blockgroupstats[,.N,by=substr(bgfips,1,2)]
length(fips_bg_from_anyfips("72"))

# all blockgroups in this one county
fips_bg_from_anyfips(30001)
fips_bg_from_anyfips("30001")

# all blockgroups that contain any of these 6 blocks (i.e., just one bg)
fips_bg_from_anyfips( blockid2fips$blockfips[1:6])

# 2 counties
fips_bg_from_anyfips(c(36009,36011))
```

`fips_counties_from_statefips`

fips_counties_from_statefips - Get ALL county fips in specified states Get all county fips in specified states

---

Description

fips_counties_from_statefips - Get ALL county fips in specified states Get all county fips in specified states

Usage

```
fips_counties_from_statefips(statefips)
```

Arguments

`statefips`	vector of 2-digit state FIPS codes like c("10", "44", "44") or c(10,44)

Details

Very similar to list_counties(state) from the tigris package.

Value

vector of 5-digit character string county FIPS of all unique counties in those states

Examples

```
fips_counties_from_statefips(c(10,44,44))
fips_counties_from_statefips("10")
```

`fips_counties_from_statename`

fips_counties_from_statename - Get ALL county fips in specified states Get all county fips in specified states

Description

fips_counties_from_statename - Get ALL county fips in specified states Get all county fips in specified states

Usage

```
fips_counties_from_statename(statename)
```

Arguments

`statename`	vector of state names like c("New York","Georgia")

Value

vector of 5-digit character string county FIPS of all unique counties in those states

Examples

```
fips_counties_from_statename("Delaware")
```

`fips_counties_from_state_abbrev`

fips_counties_from_state_abbrev - Get ALL county fips in specified states Get all county fips in specified states

Description

fips_counties_from_state_abbrev - Get ALL county fips in specified states Get all county fips in specified states

Usage

```
fips_counties_from_state_abbrev(ST)
```

Arguments

`ST`	vector of state abbreviations like c("NY","GA")

Value

vector of 5-digit character string county FIPS of all unique counties in those states

Examples

```
fips_counties_from_state_abbrev("DE")
fips_counties_from_state_abbrev("RI", "RI")
```

`fips_from_table` *fips_from_table - read and clean FIPS column from a table, after inferring which col it is Just read the codes in one column of a table obtained from something like read.csv, or excel, etc.*

Description

fips_from_table - read and clean FIPS column from a table, after inferring which col it is Just read the codes in one column of a table obtained from something like read.csv, or excel, etc.

Usage

```
fips_from_table(fips_table, addleadzeroes = TRUE, inshiny = FALSE)
```

Arguments

`fips_table`	data.frame or data.table of FIPS codes for counties, states, or tracts, for example, in a column whose name can be interpreted as FIPS (is one of the aliases like fips, countyfips, etc.) Aliases are: c("FIPS", "fips", "fips_code", "fipscode", "Fips", "statefips", "countyfips", "ST_FIPS", "st_fips", "ST_FIPS", "st_fips", "FIPS.ST", "FIPS.COUNTY", "FIPS.TRACT")
`addleadzeroes`	whether to add leading zeroes where needed as for a State whose FIPS starts with "01"
`inshiny`	used by server during shiny app

Value

a vector of fips codes

See Also

`fips_bg_from_anyfips()` `fips_lead_zero()` `getblocksnearby_from_fips()` `fips_from_table()`

`fips_lead_zero` *fips_lead_zero Add leading zeroes to fips codes if missing, replace with NA if length invalid Note it does NOT VALIDATE FIPS - It does NOT check if FIPS is valid other than checking its length seems OK, i.e., it might be a state, county, tract, blockgroup, or block FIPS code.*

Description

fips_lead_zero Add leading zeroes to fips codes if missing, replace with NA if length invalid Note it does NOT VALIDATE FIPS - It does NOT check if FIPS is valid other than checking its length seems OK, i.e., it might be a state, county, tract, blockgroup, or block FIPS code.

Usage

```
fips_lead_zero(fips)
```

Arguments

fips	vector of numeric or character US FIPS codes

Value

vector of same length

Examples

```
fips_lead_zero(c(1,"01",1234,"1234","12345",123456))
```

---

fips_st2eparegion *fips_st2eparegion - Get EPA Region number from state FIPS code Get EPA Region number from state FIPS code*

Description

fips_st2eparegion - Get EPA Region number from state FIPS code Get EPA Region number from state FIPS code

Usage

```
fips_st2eparegion(stfips)
```

Arguments

stfips	vector of one or more state fips codes (numbers or as strings)

Value

vector of numbers representing US EPA Regions

---

fips_state_from_statename *fips_state_from_statename - Get state fips for each state name Get state fips for each state name*

Description

fips_state_from_statename - Get state fips for each state name Get state fips for each state name

Usage

```
fips_state_from_statename(statename)
```

Arguments

statename	vector of state names like c("New York","Georgia")

Value

vector of 2-digit state FIPS codes like c("10", "44", "44"), same length as input, so including any duplicates

Examples

```
fips_state_from_statename("Delaware")
```

`fips_state_from_state_abbrev` *fips_state_from_state_abbrev - Get state fips for each state abbrev Get state fips of each state abbrev*

Description

fips_state_from_state_abbrev - Get state fips for each state abbrev Get state fips of each state abbrev

Usage

```
fips_state_from_state_abbrev(ST)
```

Arguments

ST	vector of state abbreviations like c("NY","GA")

Value

vector of 2-digit state FIPS codes like c("10", "44", "44"), same length as input, so including any duplicates

Examples

```
fips_state_from_state_abbrev("DE", "DE", "RI")
```

`fixcolnames2related` *fixcolnames2related - get name of related avg, pctile, or ratio variable name Given names_d, e.g., returns names_d_ratio_to_state_avg*

Description

fixcolnames2related - get name of related avg, pctile, or ratio variable name Given names_d, e.g., returns names_d_ratio_to_state_avg

Usage

```
fixcolnames2related(
  namesnow,
  relatedtype = c("usavg", "stateavg", "uspctile", "statepctile", "usratio",
    "stateratio")
)
```

Arguments

namesnow	vector of one or more basic Envt or Demog indicator variable names found in c(names_e, names_d, names_d_subgroups)
relatedtype	One of "usavg", "stateavg", "uspctile", "statepctile", "usratio", "stateratio" (but not any of the other values among unique(map_headernames$vartype) since those give ambiguous answers).

Details

Given basic variable name(s) like "pctlowinc" or names_e, see what the related variable names are for storing the US or State percentiles, averages, or ratios to averages of the given variables.

Only works for variable names among these:

c(names_e, names_d, names_d_subgroups)

Value

vector as long as namesnow (or just returns namesnow if relatedtype is invalid)

Examples

```
names_d
fixcolnames2related(names_d, 'stateratio')
names_d_ratio_to_state_avg
fixcolnames2related(names_e, "stateavg")
fixcolnames2related(names_e, "usvag")
paste0("avg.", names_e)
fixcolnames2related(names_e, "usratio")
# names_ej # does not work with this as input
# fixcolnames2related(names_ej, "uspctile") # does not return names_ej_pctile
```

---

fixmapheadernamescolname

fixmapheadernamescolname - utility to convert aliases to proper colnames of map_headernames

---

Description

fixmapheadernamescolname - utility to convert aliases to proper colnames of map_headernames

Usage

```
fixmapheadernamescolname(x)
```

Arguments

x	character vector of colnames of map_headernames, or aliases like "long"

Value

vector where aliases are replaced with actual colnames and unmatched ones left as-is

Examples

```
fixmapheadernamescolname(c('long', 'csv', 'api', 'r'))
```

`frs` *frs (DATA) EPA Facility Registry Service table of regulated sites*

Description

This is a data.table snapshot version of the EPA FRS. You can look up sites by REGISTRY_ID in frs, and get their location, etc.

Details

This dataset can be updated by a package maintainer by using frs_update_datasets() (which is not an exported function)

The definitions of active/inactive here are not quite the same as used in ECHO. See attributes(frs) to see date created, etc.

Also, EJScreen has maps of EPA-regulated facilities of a few program types, as provided here: https://www.epa.gov/ejscreen/ejscreen-map-descriptions#sites-reporting-to-epa

```
 As of November 2023

 Count of    all REGISTRY_ID rows:   7,441,086
 Count of unique REGISTRY_ID values: 4,705,744
 Clearly inactive unique IDs:        1,436,096
 Assumed   active unique IDs:        3,269,648

frs rows total:            3,456,042
frs clearly inactive IDs:  1,436,096
frs rows actives:          2,573,338
frs_by_programid rows:     3,440,036
frs_by_naics rows:           679,471
frs_by_sic rows:           1,081,742

  Classes ‘data.table’ and 'data.frame':
  Retained only these columns for this package

  $ lat           : num  18.4 18.4 18.5 18.2 18.2 ...
  $ lon           : num  -66.1 -66.1 -66.8 -67.1 -67.2 ...
 $ REGISTRY_ID  : chr  "110000307668" "110000307695" "110000307739" "110000307757" ...
 $ PRIMARY_NAME : chr  "HB FULLER COMPANY HBF PUERTO RICO" "RAMCO CHEMICALS INCORPORATED"
  $ NAICS         : chr  "325520" "" "311119" "312120" ...
  $ SIC           : chr  "2842" "2048" "2047, 2048, 2091" ...
  $ PGM_SYS_ACRNMS: chr  "NCDB:I02#19880913A2001 2, RCRAINFO:PRD090122136
```

See Also

frs_by_programid frs_by_naics frs_by_sic

`frsprogramcodes` *frsprogramcodes DATA EPA programs listed in Facility Registry Service*

Description

data.frame

	description	code
1	National Pollutant Discharge Elimination System (NPDES) (ICIS-NPDES)	NPDES
2	The Integrated Compliance Information System (ICIS) for Air (ICIS-Air)	AIR
3	The Resource Conservation and Recovery Act (RCRA) Information System	RCRAINFO
4	Risk Management Plan (RMP) facilities	RMP
5	The Safe Drinking Water Information System (SDWIS)	SFDW
6	The Superfund Enterprise Management System	SEMS
7	Clean Air Markets Division Business System	CAMDBS
8	Toxics Release Inventory Program	TRIS
9	Greenhouse Gas Reporting Program	E-GGRT
10	Emissions Inventory System	EIS
11	Toxic Substances Control Act	TSCA

Details

Created by script in /data-raw/

See Also

frs

Examples

```
## Not run:
  frs_by_programid[program %in% frsprogramcodes$code, .N, by=program]

  setkey(frs_by_programid,"program")
  frs_by_programid["TRIS",]

## End(Not run)
```

frs_by_mact — *frs_by_mact (DATA) MACT NESHAP subpart(s) that each EPA-regulated site is subject to*

Description

This is a data.table with one row per site – MACT subpart pair, so it has multiple rows for one site if the site is covered by multiple subparts. It has been joined with frs_by_programid to get latlons for matching facilities. @details

There are about 112k rows here but only about 83k unique program IDs in this table,

which is from the ECHO data download of ICIS Air and AFS.

The programid column here should be found in the pgm_sys_id column in frs_by_programid,

but as of 6/14/23 only a little over half of them were found there, so this is work in progress

to be resolved.

table(frs_by_mact$programid %in% frs_by_programid$pgm_sys_id)

FALSE TRUE

56497 55411

Also there are some typos in the downloaded dataset from ECHO/FRS, such as

"WOOD PERSERVING AREA SOURCES"

See Also

mact_table frs_by_programid frs

Examples

```
mact_table
mact_table[order(mact_table$title),]
mycodes <- c("BBBBBB", "0000")
frs_by_mact[subpart %in% mycodes, ]
mact_table[grepl("smelt", mact_table$title, ignore.case = T), ]
frs_by_mact[grepl("smelt", title, ignore.case = T), ]
# a single site can be covered by 19 categories
frs_by_mact[, howmany := .N, by="programid"][order(howmany), ]
table(frs_by_mact[, howmany := .N, by="programid"][order(howmany), howmany])
```

frs_by_naics — *frs_by_naics (DATA) data.table of NAICS code(s) for each EPA-regulated site in Facility Registry Service*

Description

This is the format with one row per site-NAICS pair, so multiple rows for one site if it is in multiple NAICS. @details

MOST SITES LACK NAICS INFO IN FRS! NAICS is missing for about 80 percent of these facilities.

frs here had about 2,571,750 unique REGISTRY_ID values, but

frs_by_naics had only about 680,000 rows as of 4/1/2023,

about 562,000 unique REGISTRY_ID values with

about 2,900 unique NAICS codes.

length(unique(frs_by_naics$REGISTRY_ID))

`length(unique(frs_by_naics[,REGISTRY_ID]))`

`length(frs_by_naics[, unique(REGISTRY_ID)])`

`frs_by_naics[,uniqueN(REGISTRY_ID)]`

```
 561,999  as of 3/26/23 but early 2023 had been 564,770


    lat       lon  REGISTRY_ID  NAICS
```

1: 34.04722 -81.15136 110000854246 325211

2: 34.04722 -81.15136 110000854246 325220

3: 34.04722 -81.15136 110000854246 325222

See Also

frs `frs_from_naics()` `naics_categories()` frs_by_programid and see naics_from_any in EJAM pkg.

Examples

```
# NAICS is missing for about 80 percent of facilities
`frs[ NAICS == "", .N] / frs[,.N] `
# only about 562k facilities have some NAICS info
`frs[ NAICS != "", .N]`
`frs_by_naics[, uniqueN(REGISTRY_ID)]` # almost exactly matches the above

dim(frs_by_naics)
# about 680k rows here, or pairs of 1 NAICS - 1 registry ID pair,
#  since some IDs have 2 or more NAICS so appear as 2 or more rows here.

# About 2,900 different NAICS codes appear here:
`frs_by_naics[,  uniqueN(NAICS)]`
`frs_by_naics[, .(sum(.N > 1)), by=NAICS][,sum(V1)]`
  #  2,457 NAICS codes are used to describe more than one Registry ID
 `frs_by_naics[, .(sum(.N == 1)), by=NAICS][,sum(V1)]`
  # [1] 425 NAICS codes appear only once, i.e., apply to only a single facility!

# Which 2-digit NAICS are found here most often?
`frs_by_naics[ , .N, keyby=substr(NAICS,1,2)]`
`frs_by_naics[ , .N,   by=substr(NAICS,1,2)][order(N),]` # Most common is 33
```

```
# Top 10 most common 3-digit NAICS here:
`x = tail(frs_by_naics[ , .N,   by=.(n3 = substr(NAICS,1,3))][order(N), ],10)`
`cbind(x, industry = rownames(naics_categories(3))[match(x$n3, naics_categories(3))])`
```

`frs_by_programid` *frs_by_programid (DATA) data.table of Program System ID code(s) for each EPA-regulated site in the Facility Registry Service*

Description

frs_by_programid (DATA) data.table of Program System ID code(s) for each EPA-regulated site in the Facility Registry Service

Details

Created by frs_make_programid_lookup() that was in EJAMfrsdata package

```
   This is the format with one row per site-programid pair,
   so multiple rows for one site if it is in multiple programs.

 > dim(frs_by_programid)
 [1] 3440451       5   as of 1/3/2023

nn=sample(1:nrow(frs_by_programid), 1); frs_by_programid[REGISTRY_ID == frs_by_programid$REGISTRY

             lat       lon  REGISTRY_ID  program         pgm_sys_id
    1: 40.21262 -100.6464 110040499724 AIRS/AFS         3114500040
    2: 40.21262 -100.6464 110040499724     NDEQ              87933
    3: 40.21262 -100.6464 110040499724      AIR NE0000003114500040

nn=sample(1:nrow(frs_by_programid), 1); frs_by_programid[REGISTRY_ID == frs_by_programid$REGISTRY

             lat       lon  REGISTRY_ID program   pgm_sys_id
    1: 47.00071 -120.5649 110037546493 WA-FSIS      1796553
    2: 47.00071 -120.5649 110037546493    ICIS   1800041945
    3: 47.00071 -120.5649 110037546493 WA-FSIS      7886103
```

See Also

frs() frs_by_naics()

`frs_by_sic` *frs_by_sic (DATA) data.table of SIC code(s) for each EPA-regulated site in Facility Registry Service*

Description

This is the format with one row per site-SIC pair, so multiple rows for one site if it is in multiple SIC.

frs_from_naics *frs_from_naics - Use NAICS code or industry title text search to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

frs_from_naics - Use NAICS code or industry title text search to see FRS Facility Registry Service data on those EPA-regulated sites

Usage

```
frs_from_naics(naics_code_or_name, ...)
```

Arguments

naics_code_or_name	
...	passed to naics_from_any()

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

See Also

siteid_from_naics() naics_from_any()

Examples

```
frs_from_naics("uranium")
mapfast(frs_from_naics(naics_from_any("nuclear")$code))
naics_from_any("silver")
naics_from_name("silver")
naics_from_any(212222 )
frs_from_naics(21222)
siteid_from_naics(21222)
latlon_from_naics(21222)
```

frs_from_program *frs_from_program - Use EPA Program acronym like TRIS to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

Get data.table based on given FRS Program System CATEGORY. Find all FRS sites in a program like RCRAINFO, TRIS, or others.

Usage

```
frs_from_program(program)
```

Arguments

program	vector of one or more EPA Program names used by FRS

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

---

frs_from_programid — *frs_from_programid - Use EPA Program ID to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

frs_from_programid - Use EPA Program ID to see FRS Facility Registry Service data on those EPA-regulated sites

Usage

```
frs_from_programid(programid)
```

Arguments

siteid	vector of one or more EPA Program ID codes used by FRS

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

Examples

```
x=frs_from_programid(testids_program_sys_id)
  x
  mapfast(x)
```

`frs_from_regid` *frs_from_siteid - Use registry ID to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

frs_from_siteid - Use registry ID to see FRS Facility Registry Service data on those EPA-regulated sites

Usage

```
frs_from_regid(siteid)
```

Arguments

`siteid`	vector of one or more EPA Registry ID codes used by FRS

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

Examples

```
frs_from_siteid(testids_registry_id)
```

`frs_from_sic` *frs_from_sic - Use SIC code or industry title text search to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

frs_from_sic - Use SIC code or industry title text search to see FRS Facility Registry Service data on those EPA-regulated sites

Usage

```
frs_from_sic(sic_code_or_name, ...)
```

Arguments

`...`	passed to `naics_from_any()`
`naics_code_or_name`	

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "SIC" "PGM_SYS_ACRNMS"

See Also

siteid_from_sic() sic_from_any()

Examples

```
frs_from_sic("glass")
mapfast(frs_from_sic(sic_from_any("silver")$code))
sic_from_any("silver")
sic_from_name("silver")
sic_from_any('0780')
frs_from_sic('0780')
siteid_from_sic('0780')
latlon_from_sic('0780')
```

---

frs_from_siteid *frs_from_siteid - Use registry ID to see FRS Facility Registry Service data on those EPA-regulated sites*

Description

frs_from_siteid - Use registry ID to see FRS Facility Registry Service data on those EPA-regulated sites

Usage

```
frs_from_siteid(siteid)
```

Arguments

siteid	vector of one or more EPA Registry ID codes used by FRS

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

Examples

```
frs_from_siteid(testids_registry_id)
```

`frs_from_sitename` *frs_from_sitename - Use site name text search to see FRS Facility Registry Service data on those EPA-regulated sites VERY SLOW search within PRIMARY_NAME of facilities for matching text*

Description

frs_from_sitename - Use site name text search to see FRS Facility Registry Service data on those EPA-regulated sites VERY SLOW search within PRIMARY_NAME of facilities for matching text

Usage

```
frs_from_sitename(sitenames, ignore.case = TRUE, fixed = FALSE)
```

Arguments

`sitenames`	one or more strings in a vector, which can be regular expressions or query for exact match using fixed=TRUE
`ignore.case`	logical, search is not case sensitive by default (unlike `grepl()` default)
`fixed`	see `grepl()`, if set to TRUE it looks for only exact matches

Value

relevant rows of the data.table called frs, which has column names that are "lat" "lon" "REGISTRY_ID" "PRIMARY_NAME" "NAICS" "PGM_SYS_ACRNMS"

Examples

```
## Not run:
 # very slow
 x=frs_from_sitename
 nrow(x)
 head(x)

## End(Not run)
```

`frs_is_valid` *frs_is_valid - Validate FRS Registry ID list uploads*

Description

Check for proper FRS facility id in uploaded data

Usage

```
frs_is_valid(frs_upload)
```

Arguments

`frs_upload`	upload frs registry IDs table converted to data frame

Value

boolean value (valid or not valid)

`getblocksnearby` *getblocksnearby - Fast way to find nearby points (distance to each Census block centroid near each site)*

Description

Given a set of points and a specified radius, this function quickly finds all the US Census blocks near each point. For each point, it uses the specified radius distance and finds the distance to every block within the circle defined by the radius. Each block is defined by its Census-provided internal point, by latitude and longitude.

Each point can be the location of a regulated facility or other type of site, and the blocks are a high-resolution source of information about where residents live.

Finding which blocks have their internal points in a circle provides a way to quickly estimate what fraction of a block group is inside the circular buffer more accurately and more quickly than areal apportionment of block groups would provide.

Usage

```
getblocksnearby(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = FALSE,
  quadtree,
  quiet = FALSE,
  parallel = FALSE,
  ...
)
```

Arguments

`sitepoints`	data.table with columns siteid, lat, lon giving point locations of sites or facilities around which are circular buffers
`radius`	in miles, defining circular buffer around a site point
`maxradius`	miles distance (max distance to check if not even 1 block point is within radius)
`avoidorphans`	logical If TRUE, then where not even 1 BLOCK internal point is within radius of a SITE, it keeps looking past radius, up to maxradius, to find nearest 1 BLOCK. What EJScreen does in that case is report NA, right? So, does EJAM really need to report stats on residents presumed to be within radius, if no block centroid is within radius? Best estimate might be to report indicators from nearest block centroid which is probably almost always the one your site is sitting inside of, but ideally would adjust total count to be a fraction of blockwt based on what is area of circular buffer as fraction of area of block it is apparently inside of. Setting this to TRUE can produce unexpected results, which will not match EJScreen numbers. Note that if creating a proximity score, by contrast, you instead want to find nearest 1 SITE if none within radius of this BLOCK.

quadtree	(a pointer to the large quadtree object) created using indexblocks() which uses the SearchTree package. Takes about 2-5 seconds to create this each time it is needed. It can be automatically created when the package is attached via the .onAttach() function
quiet	Optional. set to TRUE to avoid message about using getblock_diagnostics(), which is relevant only if a user saved the output of this function.
...	passed to getblocksnearbyviaQuadTree() or other such functions

Details

See ejamit() for examples.

getblocksnearby() is a wrapper redirecting to the right version, like getblocksnearbyviaQuadTree() Census block "internal points" (defined by Census Bureau) are actually what it looks for, and they are like centroids. The blocks are pre-indexed for the whole USA, via the data object quadtree aka localtree

See Also

ejamit() getblocksnearbyviaQuadTree() getblocksnearbyviaQuadTree_Clustered() getblocksnearbyviaQua

---

getblocksnearby2 *getblocksnearby2 - Key buffering function - wrapper redirecting to the right version of getblocksnearby() DRAFT / WAS WORK IN PROGRESS*

Description

getblocksnearby2 - Key buffering function - wrapper redirecting to the right version of getblocksnearby() DRAFT / WAS WORK IN PROGRESS

Usage

```
getblocksnearby2(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = FALSE,
  quadtree = is.null,
  ...
)
```

Arguments

sitepoints	see getblocksnearbyviaQuadTree() or other such functions
radius	see getblocksnearbyviaQuadTree() or other such functions
maxradius	see getblocksnearbyviaQuadTree() or other such functions
avoidorphans	see getblocksnearbyviaQuadTree() or other such functions
quadtree	a large quadtree object created from the SearchTree package example: SearchTrees::createTree(quaddata, treeType = "quad", dataType = "point")
...	see getblocksnearbyviaQuadTree_Clustered() or other such functions

Details

For all examples, see `ejamit()`

Like getblocksnearby() but tries to handle localtree and quadtree parameter differently

- not sure how to check if they are in the right environment.

See Also

`getblocksnearby()`

`getblocksnearbyviaQuadTree`

getblocksnearbyviaQuadTree - Fast way to find nearby points (distance to each Census block centroid near each site)

Description

Given a set of points and a specified radius in miles, this function quickly finds all the US Census blocks near each point.

Usage

```
getblocksnearbyviaQuadTree(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = FALSE,
  report_progress_every_n = 500,
  quiet = FALSE,
  retain_unadjusted_distance = TRUE,
  quadtree
)
```

Arguments

`sitepoints`	data.table with columns siteid, lat, lon giving point locations of sites or facilities around which are circular buffers
`radius`	in miles, defining circular buffer around a site point
`maxradius`	miles distance (max distance to check if not even 1 block point is within radius)
`avoidorphans`	logical If TRUE, then where not even 1 BLOCK internal point is within radius of a SITE, it keeps looking past radius, up to maxradius, to find nearest 1 BLOCK. What EJScreen does in that case is report NA, right? So, does EJAM really need to report stats on residents presumed to be within radius, if no block centroid is within radius? Best estimate might be to report indicators from nearest block centroid which is probably almost always the one your site is sitting inside of, but ideally would adjust total count to be a fraction of blockwt based on what is area of circular buffer as fraction of area of block it is apparently inside of. Setting this to TRUE can produce unexpected results, which will not match EJScreen numbers. Note that if creating a proximity score, by contrast, you instead want to find nearest 1 SITE if none within radius of this BLOCK.

`report_progress_every_n`
: Reports progress to console after every n points, mostly for testing, but a progress bar feature might be useful unless this is super fast.

`quiet`
: Optional. set to TRUE to avoid message about using getblock_diagnostics(), which is relevant only if a user saved the output of this function.

`retain_unadjusted_distance`
: set to FALSE to drop it and save memory/storage. If TRUE, the distance_unadjusted column will save the actual distance of site to block internal point – the distance column always represents distance to average resident in the block, which is estimated by adjusting the site to block distance in cases where it is small relative to the size of the block, to put a lower limit on it, which can result in a large estimate of distance if the block is very large. See EJScreen documentation.

`quadtree`
: (a pointer to the large quadtree object) created using indexblocks() which uses the SearchTree package. Takes about 2-5 seconds to create this each time it is needed. It can be automatically created when the package is attached via the .onAttach() function

Details

For each point, it uses the specified search radius and finds the distance to every block within the circle defined by the radius. Each block is defined by its Census-provided internal point, by latitude and longitude.

Results are the sites2blocks table that would be used by doaggregate(), with distance in miles as one output column of data.table. Adjusts distance to avg resident in block when it is very small relative to block size, the same way EJScreen adjusts distances in creating proximity scores.

Each point can be the location of a regulated facility or other type of site, and the blocks are a high-resolution source of information about where residents live.

Finding which blocks have their internal points in a circle provides a way to quickly estimate what fraction of a block group is inside the circular buffer more accurately and more quickly than areal apportionment of block groups would provide.

See Also

`ejamit()` `getblocksnearby()`

Examples

```
# indexblocks() # if localtree not available yet, quadtree = localtree
x = getblocksnearby(testpoints_1000, radius = 3)
```

---

getblocksnearbyviaQuadTree2

getblocksnearbyviaQuadTree2 - Find nearby blocks using Quad Tree data structure for speed, NO PARALLEL PROCESSING - DRAFT / WORK IN PROGRESS

Description

Given a set of points and a specified radius in miles, this function quickly finds all the US Census blocks near each point.

Usage

```
getblocksnearbyviaQuadTree2(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = FALSE,
  report_progress_every_n = 500,
  quiet = FALSE,
  quadtree
)
```

Arguments

sitepoints	data.table with columns siteid, lat, lon giving point locations of sites or facilities around which are circular buffers
radius	in miles, defining circular buffer around a site point
maxradius	miles distance (max distance to check if not even 1 block point is within radius)
avoidorphans	logical If TRUE, then where not even 1 BLOCK internal point is within radius of a SITE, it keeps looking past radius, up to maxradius, to find nearest 1 BLOCK. What EJScreen does in that case is report NA, right? So, does EJAM really need to report stats on residents presumed to be within radius, if no block centroid is within radius? Best estimate might be to report indicators from nearest block centroid which is probably almost always the one your site is sitting inside of, but ideally would adjust total count to be a fraction of blockwt based on what is area of circular buffer as fraction of area of block it is apparently inside of. Setting this to TRUE can produce unexpected results, which will not match EJScreen numbers. Note that if creating a proximity score, by contrast, you instead want to find nearest 1 SITE if none within radius of this BLOCK.
report_progress_every_n	Reports progress to console after every n points, mostly for testing, but a progress bar feature might be useful unless this is super fast.
quiet	Optional. set to TRUE to avoid message about using getblock_diagnostics(), which is relevant only if a user saved the output of this function.
quadtree	(a pointer to the large quadtree object) created using indexblocks() which uses the SearchTree package. Takes about 2-5 seconds to create this each time it is needed. It can be automatically created when the package is attached via the .onAttach() function

Details

This should be almost identical to getblocksnearbyviaQuadTree(), but it uses f2, a copy of sitepoints, and more importantly pulls some code out of the for loop and uses a vectorized approach. For each point, it uses the specified search radius and finds the distance to every block within the circle defined by the radius. Each block is defined by its Census-provided internal point, by latitude and longitude.

Results are the sites2blocks table that would be used by doaggregate(), with distance in miles as one output column of data.table. Adjusts distance to avg resident in block when it is very small relative to block size, the same way EJScreen adjusts distances in creating proximity scores.

Each point can be the location of a regulated facility or other type of site, and the blocks are a high-resolution source of information about where residents live.

Finding which blocks have their internal points in a circle provides a way to quickly estimate what fraction of a block group is inside the circular buffer more accurately and more quickly than areal apportionment of block groups would provide.

See Also

ejamit() getblocksnearby()

Examples

```
# indexblocks() # if localtree not available yet, quadtree = localtree
x = getblocksnearby2(testpoints_1000, radius = 3)
```

---

getblocksnearbyviaQuadTree3

getblocksnearbyviaQuadTree3 - Find nearby blocks using Quad Tree data structure for speed, NO PARALLEL PROCESSING DRAFT / WAS WORK IN PROGRESS

Description

Given a set of points and a specified radius (in miles), this function quickly finds all the US Census blocks near each point. For each point, it uses the specified search radius and finds the distance to every block within the circle defined by the radius. Each block is defined by its Census-provided internal point, by latitude and longitude.

Each point can be the location of a regulated facility or other type of site, and the blocks are a high-resolution source of information about where residents live.

Finding which blocks have their internal points in a circle provides a way to quickly estimate what fraction of a block group is inside the circular buffer more accurately and more quickly than areal apportionment of block groups would provide.

Usage

```
getblocksnearbyviaQuadTree3(
  sitepoints,
  radius = 3,
  maxradius = 31.07,
  avoidorphans = TRUE,
  report_progress_every_n = 500,
  quadtree
)
```

Arguments

sitepoints	data.table with columns siteid, lat, lon giving point locations of sites or facilities around which are circular buffers
radius	in miles, defining circular buffer around site point
maxradius	miles distance (max distance to check if not even 1 block point is within radius)
avoidorphans	logical Whether to avoid case where no block points are within radius, so if TRUE, it keeps looking past radius to find nearest one within maxradius.

`report_progress_every_n`
: Reports progress to console after every n points, mostly for testing, but a progress bar feature might be useful unless this is super fast.

`quadtree`
: (a pointer to the large quadtree object) created from the SearchTree package example: SearchTrees::createTree(quaddata, treeType = "quad", dataType = "point") Takes about 2-5 seconds to create this each time it is needed. It is automatically created when the package is attached via the .onAttach() function

See Also

`ejamit()` `getblocksnearby()` `getblocksnearbyviaQuadTree()` `getblocksnearbyviaQuadTree_Clustered()` `getblocksnearbyviaQuadTree2()`

`getblocksnearbyviaQuadTree_Clustered`
: *getblocksnearbyviaQuadTree_Clustered - find nearby blocks using Quad Tree data structure for speed, CLUSTERED FOR PARALLEL PROCESSING*

Description

Uses packages parallel and snow. parallel::makePSOCKcluster is an enhanced version of snow::makeSOCKcluster in package snow. It runs Rscript on the specified host(s) to set up a worker process which listens on a socket for expressions to evaluate, and returns the results (as serialized objects).

Usage

```
getblocksnearbyviaQuadTree_Clustered(
  sitepoints,
  radius,
  maxradius,
  avoidorphans,
  CountCPU = 1,
  quadtree
)
```

Arguments

`sitepoints`	data.table with columns LAT, LONG
`radius`	in miles
`maxradius`	miles distance
`avoidorphans`	logical
`CountCPU`	for parallel processing via makeCluster() and `doSNOW::registerDoSNOW()`
`quadtree`	index of all US blocks like localtree

Details

For all examples, see `getblocksnearbyviaQuadTree()`

Uses indexgridsize and quaddata variables that come from global environment (but should pass to this function rather than assume in global env?)

See Also

getblocksnearby() getblocksnearbyviaQuadTree() getblocksnearbyviaQuadTree_Clustered() getblocksnearbyviaQuadTree2()

---

getblocksnearby_from_fips *getblocksnearby_from_fips Actually finds all blocks within each of the FIPS codes provided*

Description

getblocksnearby_from_fips Actually finds all blocks within each of the FIPS codes provided

Usage

```
getblocksnearby_from_fips(fips, inshiny = FALSE, need_blockwt = TRUE)
```

Arguments

fips	vector of FIPS codes identifying blockgroups, tracts, counties, or states. This is useful if – instead of gettings stats on and comparing circular buffers or polygons – one will be getting stats on one or more tracts, or analyzing and comparing blockgroups in a county, or comparing whole counties to each other, within a State.
inshiny	used by shiny app server code to handle errors via validate() instead of stop()
need_blockwt	set to FALSE to speed it up if you do not need blockwt

Value

same as for getblocksnearby but one row per FIPS, and the distance column is irrelevant

See Also

fips_bg_from_anyfips() fips_lead_zero() getblocksnearby_from_fips() fips_from_table()

Examples

```
 x <- getblocksnearby_from_fips(fips_counties_from_state_abbrev("DE"))
 counties_ej <- doaggregate(x)
# mapfast(counties_ej$results_bysite)
 y =  ejamit(fips=fips_counties_from_statename("Delaware"))

 # x=getblocksnearby_from_fips("482011000011") # one blockgroup only
 # y=doaggregate(x)
```

getblocks_diagnostics *getblocks_diagnostics - Lots of stats about # of blocks, # of sites, etc.*

Description

getblocks_diagnostics - Lots of stats about # of blocks, # of sites, etc.

Usage

```
getblocks_diagnostics(x, detailed = FALSE, see_pctiles = FALSE)
```

Arguments

x	The output of getblocksnearby() like testoutput_getblocksnearby_10pts_1miles
detailed	if TRUE, also shows in console a long table of frequencies via getblocks_summarize_blocks_per_s
see_pctiles	set to TRUE to see 20 percentiles of distance in a table

Value

A list of stats

See Also

This relies on getblocks_summarize_blocks_per_site() and getblocks_summarize_sites_per_block()

Examples

```
 getblocks_diagnostics(testoutput_getblocksnearby_10pts_1miles)
 # library(data.table)
 x <- data.table::copy(testpoints_10)
 setDT(x)
 pts <- rbind(data.table(lat = 40.3, lon = -96.23),
   x[ , .(lat, lon)])
z <- getblocksnearbyviaQuadTree(pts, 1, quadtree = localtree, quiet = T)
z[ , .(blocks = .N) , keyby = 'siteid']
plotblocksnearby(pts, radius = 1, sites2blocks = z)
zz <- getblocks_diagnostics(z, detailed = T, see_pctiles = T)
cbind(stats = zz)
```

---

getblocks_summarize_blocks_per_site *getblocks_summarize_blocks_per_site - how many blocks are near the sites (pop density affects accuracy)*

Description

Number of blocks near avg site, how many sites have only 1 or fewer than 30 blocks nearby, etc.

Usage

```
getblocks_summarize_blocks_per_site(x, varname = "siteid")
```

Arguments

x	The output of getblocksnearby()
varname	colname of variable in data.table x that is the one to summarize by

Value

invisibly, a list of stats

See Also

getblocks_diagnostics()

---

getblocks_summarize_sites_per_block

getblocks_summarize_sites_per_block - how many sites are near the blocks (site density near residents)

Description

getblocks_summarize_sites_per_block - how many sites are near the blocks (site density near residents)

Usage

```
getblocks_summarize_sites_per_block(x, varname = "blockid")
```

Arguments

x	The output of getblocksnearby() like testoutput_getblocksnearby_10pts_1miles
varname	colname of variable in data.table x that is the one to summarize by

Value

invisibly, a list of stats

See Also

getblocks_diagnostics()

`get_blockpoints_in_shape` *get_blockpoints_in_shape - find blocks that are in a polygon, using internal point of block - WORK IN PROGRESS ****

Description

This is like getblocksnearby() but for a polygonal buffer area instead of a circular buffer.

Usage

```
get_blockpoints_in_shape(
  polys,
  addedbuffermiles = 0,
  blocksnearby = NULL,
  dissolved = FALSE,
  safety_margin_ratio = 1.1,
  crs = 4269
)
```

Arguments

`polys`	Spatial data as from sf::st_as_sf(), with a column called siteid, like points as from `shapefile_from_sitepoints()`, or a table of points with lat,lon columns that will first be converted here using that function, or polygons
`addedbuffermiles`	width of optional buffering to add to the points (or edges), in miles
`blocksnearby`	optional table of blocks with blockid,siteid (from which lat,lon can be looked up in blockpoints dt)
`dissolved`	If TRUE, use sf::st_union(polys) to find unique blocks inside any one or more of polys
`safety_margin_ratio`	multiplied by addedbuffermiles, how far to search for blocks nearby using get-blocksnearby(), before using those found to do the intersection via sf::
`crs`	used in st_as_sf() and st_transform() and shape_buffered_from_shapefile_points(), crs = 4269 or Geodetic CRS NAD83

Details

This uses getblocksnearby() to get a very fast rough/good estimate of which US block points are nearby (with a safety margin - see param below), before then using sf:: to carefully identify which of those candidate blocks are actually inside each polygon (e.g., circle) according to sf:: methods.

For circular buffers, just using getblocksnearby() should work and not need this function.

For noncircular polygons, buffered or not, this function will provide a way to very quickly filter down to which of the millions of US blocks should be examined by the sf:: join / intersect, since otherwise it takes forever for sf:: to check all US blocks.

Value

Block points table for those blocks whose internal point is inside the buffer which is just a circular buffer of specified radius if polys are just points.

See Also

get_blockpoints_in_shape() shapefile_from_sitepoints() shape_buffered_from_shapefile_points()

Examples

```
# y <- get_blockpoints_in_shape()

# x = shapefile_from_sitepoints(testpoints_n(2))
# y = get_blockpoints_in_shape(x, 1)  # very very slow
```

---

high_pctiles_tied_with_min

high_pctiles_tied_with_min (DATA) internal data used to handle cases where multiple places are tied for the lowest indicator score

---

Description

high_pctiles_tied_with_min (DATA) internal data used to handle cases where multiple places are tied for the lowest indicator score

---

indexblocks

indexblocks Create localtree (a quadtree index of all US block centroids) in global environment

---

Description

indexblocks Create localtree (a quadtree index of all US block centroids) in global environment

Usage

```
indexblocks()
```

Details

Note this is duplicated code in .onAttach() and also in global.R

.onAttach() can be edited to create this when the package loads, but then it takes time each time a developer rebuilds/installs the package or others that load EJAM.

It also has to happen in global.R if it has not already.

Value

Returns TRUE when done. Side effect is it creates the index in memory.

`input_names_listing` *input_names_listing Utility checking values of input$ that appear in this code See appsilon pkg shiny.info now*

Description

input_names_listing Utility checking values of input$ that appear in this code See appsilon pkg shiny.info now

Usage

```
input_names_listing(file = "./R/app_server.R")
```

Arguments

`file`	path to source file to search in

Value

character vector of ids of inputs like x,y,z if it found input$x input$y input$z

`islandareas` *islandareas (DATA) table, bounds info on lat lon of US Island Areas*

Description

data.frame of info on approximate lat lon bounding boxes around American Samoa, Guam, the Commonwealth of the Northern Mariana Islands (Northern Mariana Islands), and the United States Virgin Islands.

See also stateinfo and []

See http://www.census.gov/geo/reference/gtc/gtc_island.html

See datacreate_islandareas.R or data-raw/islandareas.xlsx

Note the US minor outlying islands are not in that list and are widely dispersed. They include Midway Islands, etc.

`latlon_as.numeric` *latlon_as.numeric - Strip non-numeric characters from a vector*

Description

Remove all characters other than minus signs, decimal points, and numeric digits

Usage

```
latlon_as.numeric(x)
```

Arguments

x	vector of something that is supposed to be numbers like latitude or longitude and may be a character vector because there were some other characters like tab or space or percent sign or dollar sign

Details

Useful if latitude or longitude vector has spaces, tabs, etc. CAUTION - Assumes stripping those out and making it numeric will fix whatever problem there was and end result is a valid set of numbers. Inf etc. are turned into NA values. Empty zero length string is turned into NA without warning. NA is left as NA. If anything other than empty or NA could not be interpreted as a number, it returns NA for those and offers a warning.

Value

numeric vector same length as x

See Also

latlon_df_clean() latlon_infer() latlon_is.valid() latlon_as.numeric()

Examples

```
latlon_as.numeric(c("-97.179167000000007", " -94.0533", "-95.152083000000005"))
latlon_as.numeric(-3:3)
latlon_as.numeric(c(1:3, NA))
latlon_as.numeric(c(1, 'asdf'))
latlon_as.numeric(c(1, ''))
latlon_as.numeric(c(1, '', NA))
latlon_as.numeric(c('aword', '$b'))
latlon_as.numeric(c('-10.5%', '<5', '$100'))
latlon_as.numeric(c(Inf, 1))
```

`latlon_df_clean` *latlon_df_clean - Find and clean up latitude and longitude columns in a data.frame*

Description

Utility to identify lat and lon columns, renaming and cleaning them up.

Usage

```
latlon_df_clean(df)
```

Arguments

`df`	data.frame With columns lat and lon or names that can be interpreted as such

Details

Tries to figure out which columns seem to have lat lon values, renames those in the data.frame. Cleans up lat and lon values (removes extra characters, makes numeric)

Value

Returns the same data.frame but with relevant colnames changed to lat and lon, and invalid lat or lon values cleaned up if possible or else replaced with NA

See Also

Used by `latlon_from_anything()`. Uses `latlon_infer()` `latlon_is.valid()` `latlon_as.numeric()`

Examples

```
#  x <- latlon_df_clean(x)
```

`latlon_from_anything` *latlon_from_anything - Flexibly get lat/lon from file, data.frame, data.table, or lat/lon vectors*

Description

Try to figure out if user provided latitude / longitude as vectors, data.frame, file, or interactively pick file.

Usage

```
latlon_from_anything(x, y)
```

Arguments

x	If missing and interactive mode in RStudio, prompts user for file. Otherwise, this can be a filename (csv or xlsx, with path), or data.frame/ data.table/ matrix, or vector of longitudes (in which case y must be the latitudes). Note that even though it is called latlon_etc the lon is x and comes before the lat among parameters x,y (unlike in most other functions here using lat,lon) File or data.frame/data.table/matrix must have columns called lat and lon, or names that can be inferred to be that by latlon_infer()
y	If x is a vector of longitudes, y must be the latitudes. Ignored otherwise.

Details

This function

relies on

read_csv_or_xl() and

`latlon_df_clean()` which in turn uses `latlon_infer()` `latlon_as.numeric()` `latlon_is.valid()`

read_and_clean_points() from EJAMejscreenapi

would be the most general / flexible broadest way to get points, but is still work in progress

is similar to what is done by latlon_from_anything()

except it also uses these functions:

`latlon_from_siteid()`

`latlon_from_programid()` but not _from_naics() ?

Value

A data.frame that has at least columns lon and lat (and others if they were in x)

See Also

read_csv_or_xl() latlon_df_clean()

Examples

```
if (interactive()) {
pts <- latlon_from_anything()
}
latlon_from_anything(testpoints_100[1:6,] )
latlon_from_anything(testpoints_100[1:6, c('lat','lon')] )
latlon_from_anything(x=testpoints_100$lon[1:6], y=testpoints_100$lat[1:6] )
```

`latlon_from_mactsubpart` *latlon_from_mactsubpart Get point locations for US EPA-regulated facilities that have sources subject to Maximum Achievable Control Technology (MACT) standards under the Clean Air Act.*

Description

latlon_from_mactsubpart Get point locations for US EPA-regulated facilities that have sources subject to Maximum Achievable Control Technology (MACT) standards under the Clean Air Act.

Usage

```
latlon_from_mactsubpart(subpart, include_if_no_latlon = FALSE)
```

Arguments

`subpart`	vector of one or more strings indicating the Subpart of CFR Title 40 Part 63 that covers the source category of interest, such as "FFFF" - see for example, https://www.ecfr.gov/current/title-40/part-63/subpart-FFFF
`include_if_no_latlon`	logical - many in the database lack lat lon values but have a MACT code

Details

See https://www.epa.gov/stationary-sources-air-pollution/national-emission-standards-hazardous-a

Value

a table of lat, lon, subpart, etc. for US EPA FRS sites with that MACT code

`latlon_from_naics` *latlon_from_naics - Find EPA-regulated facilities in FRS by NAICS code (industrial category) Get lat lon, Registry ID, given NAICS industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact NAICS code (not subcategories)*

Description

latlon_from_naics - Find EPA-regulated facilities in FRS by NAICS code (industrial category) Get lat lon, Registry ID, given NAICS industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact NAICS code (not subcategories)

Usage

```
latlon_from_naics(naics, id_only = FALSE)
```

Arguments

`naics`	a vector of naics codes, or a data.table with column named code, as with output of `naics_from_any()`

Details

NOTE: many FRS sites lack NAICS code!

Also, this function does not find the sites identified by FRS data as being in a child NAICS (subcategory of your exact query)!

Relies on frs_by_naics (a data.table)

See info about NAICS industry codes at https://www.naics.com/search

Value

A data.table (not just data.frame) with columns called lat, lon, REGISTRY_ID, NAICS (but see the id_only parameter)

Examples

```
siteid_from_naics(321114)
latlon_from_naics(321114)
latlon_from_naics(EJAM::naics_from_any("cheese")[,code] )
head(latlon_from_naics(c(3366, 33661, 336611), id_only=TRUE))
# mapfast(frs_from_naics(336611)) # simple map
```

---

latlon_from_program *latlon_from_program - Get lat lon, Registry ID, and NAICS, for given FRS Program System CATEGORY Find all FRS sites in a program like RCRAINFO, TRIS, or others*

Description

latlon_from_program - Get lat lon, Registry ID, and NAICS, for given FRS Program System CATEGORY Find all FRS sites in a program like RCRAINFO, TRIS, or others

Usage

```
latlon_from_program(query)
```

Arguments

program	like "RMP", "RCRAINFO", "TRIS", "RMP", or others.

Details

For info on FRS program codes in general, see https://www.epa.gov/frs/frs-program-crosswalks

Also see information at https://echo.epa.gov/tools/data-downloads/frs-download-summary about the file FRS_PROGRAM_LINKS.csv

For info on program codes ECHO uses, see https://echo.epa.gov/resources/echo-data/about-the-data

including https://www.epa.gov/frs/frs-environmental-interest-types

For a list of program acronyms, https://www.epa.gov/frs/frs-rest-services#appendixa

The acronym is the abbreviated name that represents the name of an information management system for an environmental program. The Federal ones with at least 100k facilities each are

RCRAINFO (over 500k sites), NPDES, ICIS, AIR, FIS, EIS, and AIRS/AFS.

Value

data.table with lat lon REGISTRY_ID program – but not pgm_sys_id since there could be duplicates where same REGISTRY_ID has 2 different pgm_sys_id values in the same program, so results were sometimes longer than if using `frs_from_program()`

Examples

```
## Not run:
 x = latlon_from_program("CAMDBS")
 EJAMejscreenapi::mapfast(x)
 program <- c("EIS", "UST")
 x = latlon_from_program(program)
 # to get the facility name as well:
 x = frs[grepl("RCRAINFO", PGM_SYS_ACRNMS), ] # fast
 ## x = latlon_from_siteid(latlon_from_program(program)[,REGISTRY_ID])  # slower!
 EJAMejscreenapi::mapfast(x[sample(1:nrow(x), 1000), ])

## End(Not run)
```

---

latlon_from_programid *latlon_from_programid - Get lat lon, Registry ID, and NAICS, for given FRS Program System ID*

Description

latlon_from_programid - Get lat lon, Registry ID, and NAICS, for given FRS Program System ID

Usage

```
latlon_from_programid(programid)
```

Arguments

programid	like "XJW000012435"

Details

The ID is the identification number, such as the permit number, assigned by an information management system that represents a facility site, waste site, operable unit, or other feature tracked by that Environmental Information System.

Also note the FRS API: https://www.epa.gov/frs/facility-registry-service-frs-api https://www.epa.gov/frs/frs-rest-services

Value

data.table with lat lon REGISTRY_ID program pgm_sys_id

Examples

```
latlon_from_programid(c("XJW000012435", "00768SRTRSROAD1"))
pids <- c("7-0540-00003", "354362", "1513529", "485659", "LAG750956",
  "CAC002995519", "3601252181", "3601439158")
latlon_from_siteid(latlon_from_programid(pids)[,REGISTRY_ID])
latlon_from_programid(c("XJW000012435", "00768SRTRSROAD1", "asdfsdf"))[,.(lat,lon)]
```

---

latlon_from_regid *latlon_from_siteid - Get lat lon (and NAICS) via Facility Registry ID*

Description

latlon_from_siteid - Get lat lon (and NAICS) via Facility Registry ID

Usage

```
latlon_from_regid(siteid)
```

Arguments

siteid	Facility Registry Service ID like 110010052520

Value

data.table with columns lat,lon,REGISTRY_ID,PRIMARY_NAME,NAICS,PGM_SYS_ACRNMS

Examples

```
 latlon_from_siteid(110070874073)
x = latlon_from_siteid(
    c(110071293460, 110070874073, 110070538057, 110044340807,
       110030509215, 110019033810, 110056111559, 110056982323)
        )
EJAMejscreenapi::mapfast(x)
```

---

latlon_from_sic *latlon_from_sic - Find EPA-regulated facilities in FRS by SIC code (industrial category)*

Description

Get lat lon, Registry ID, given SIC industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact SIC code (not subcategories)

Usage

```
latlon_from_sic(sic, id_only = FALSE)
```

Arguments

sic	a vector of SIC codes, or a data.table with column named code, as with output of sic_from_any()

Details

NOTE: many FRS sites lack SIC code!

Also, this function does not find the sites identified by FRS data as being in a child SIC (subcategory of your exact query)!

Relies on frs_by_sic (a data.table)

See info about SIC industry codes at https://www.naics.com/search

Value

A data.table (not just data.frame) with columns called lat, lon, REGISTRY_ID, SIC (but see the id_only parameter)

Examples

```
siteid_from_sic('7300')
latlon_from_sic('7300')
latlon_from_sic(sic_from_any("cheese")[,code] )
head(latlon_from_sic(c('6150', '6300', '5995'), id_only=TRUE))
# mapfast(frs_from_sic('6150')) # simple map
```

---

latlon_from_siteid *latlon_from_siteid - Get lat lon (and NAICS) via Facility Registry ID*

Description

latlon_from_siteid - Get lat lon (and NAICS) via Facility Registry ID

Usage

```
latlon_from_siteid(siteid)
```

Arguments

siteid	Facility Registry Service ID like 110010052520

Value

data.table with columns lat,lon,REGISTRY_ID,PRIMARY_NAME,NAICS,PGM_SYS_ACRNMS

Examples

```
 latlon_from_siteid(110070874073)
x = latlon_from_siteid(
    c(110071293460, 110070874073, 110070538057, 110044340807,
       110030509215, 110019033810, 110056111559, 110056982323)
        )
EJAMejscreenapi::mapfast(x)
```

latlon_infer — *latlon_infer - guess which columns have lat and lon based on aliases like latitude, FacLat, etc.*

Description

latlon_infer - guess which columns have lat and lon based on aliases like latitude, FacLat, etc.

Usage

```
latlon_infer(mycolnames)
```

Arguments

mycolnames	e.g., colnames(x) where x is a data.frame from read.csv

Value

returns all of mycolnames except replacing the best candidates with lat and lon

See Also

latlon_df_clean() latlon_infer() latlon_is.valid() latlon_as.numeric()

Examples

```
latlon_infer(c('trilat', 'belong', 'belong')) # warns if no alias found,
  #  but doesnt warn of dupes in other terms, just preferred term.
latlon_infer(c('a', 'LONG', 'Longitude', 'lat')) # only the best alias is converted/used
latlon_infer(c('a', 'LONGITUDE', 'Long', 'Lat')) # only the best alias is converted/used
latlon_infer(c('a', 'longing', 'Lat', 'lat', 'LAT')) # case variants of preferred are
    # left alone only if lowercase one is found
latlon_infer(c('LONG', 'long', 'lat')) # case variants of a single alias are
    # converted to preferred word (if pref not found), creating dupes!  warn!
latlon_infer(c('LONG', 'LONG')) # dupes of an alias are renamed and still are dupes! warn!
latlon_infer(c('lat', 'lat', 'Lon')) # dupes left as dupes but warn!
```

latlon_is.available — *latlon_is.available check if not NA using !is.na()*

Description

latlon_is.available check if not NA using !is.na()

Usage

```
latlon_is.available(lat, lon)
```

Arguments

lat	vector of latitudes
lon	vector of longitudes

Value

logical vector, one element per lat lon pair (location)

See Also

latlon_is.usa() latlon_is.islandareas() latlon_is.available() latlon_is.possible()

---

latlon_is.islandareas *latlon_is.islandareas Check lat lon coordinates to see if each is approx. in general area of US Island Areas Guam, USVI, Amer Samoa or N Marianas See islandareas*

Description

latlon_is.islandareas Check lat lon coordinates to see if each is approx. in general area of US Island Areas Guam, USVI, Amer Samoa or N Marianas See islandareas

Usage

```
latlon_is.islandareas(lat, lon)
```

Arguments

lat	vector of latitudes
lon	vector of longitudes

Value

vector of TRUE / FALSE values indicating a given lat lon pair is approximately in one of the rough bounding boxes that includes the 4 Island Areas.

See Also

latlon_is.usa() latlon_is.islandareas() latlon_is.available() latlon_is.possible()

Examples

```
## Not run:
# this would require the testpoints_1000 data from the EJAM package:
 isles <- which(latlon_is.islandareas(lat = testpoints_1000$lat, lon = testpoints_1000$lon))
  mapfast(testpoints_1000[isles, ]) # c(213,785)
  which(!(latlon_is.usa(lat = testpoints_1000$lat, lon = testpoints_1000$lon)))

## End(Not run)
```

latlon_is.possible *latlon_is.possible check if between -180 and +180*

Description

latlon_is.possible check if between -180 and +180

Usage

```
latlon_is.possible(lat, lon)
```

Arguments

lat	vector of latitudes
lon	vector of longitudes

Value

logical vector, one element per lat lon pair (location)

See Also

latlon_is.usa() latlon_is.islandareas() latlon_is.available() latlon_is.possible()

latlon_is.usa *latlon_is.usa Check lat lon coordinates to see if each is approx. in general area of USA excluding Island Areas*

Description

latlon_is.usa Check lat lon coordinates to see if each is approx. in general area of USA excluding Island Areas

Usage

```
latlon_is.usa(lat, lon)
```

Arguments

lat	vector of latitudes
lon	vector of longitudes

Value

logical vector, one element per lat lon pair (location) Indicates the point is approximately in one of the rough bounding boxes that includes the USA without the Island Areas Guam, American Samoa, USVI, N Marianas Islands.

See Also

latlon_is.usa() latlon_is.islandareas() latlon_is.available() latlon_is.possible()

`latlon_is.valid` *latlon_is.valid - Validate latitudes and longitudes*

Description

Check each latitude and longitude value to see if they are valid.

Usage

```
latlon_is.valid(lat, lon, quiet = TRUE)
```

Arguments

`lat`	vector of latitudes
`lon`	vector of longitudes
`quiet`	optional logical, if TRUE, show list of bad values in console

Details

NA or outside expected numeric ranges

(based on approx ranges of lat lon seen among block internal points dataset)

But note Guam, American Samoa, Northern Mariana Islands, and U.S. Virgin Islands are outside these ranges! EJScreen 2.2 does not provide demographic data in those locations anyway, but can map sites there.

lat must be between 17.5 and 71.5, and

lon must be (between -180 and -64) OR (between 172 and 180)

Value

logical vector, one element per lat lon pair (location)

See Also

`latlon_is.usa()` `latlon_is.islandareas()` `latlon_is.available()` `latlon_is.possible()` `latlon_df_clean()` `latlon_infer()` `latlon_is.valid()` `latlon_as.numeric()`

Examples

```
 ## Not run:
 # this would only work using the EJAM package datasets frs and blockpoints:
   if (!exists("frs")) dataload_from_pins("frs")
 table(latlon_is.valid(lat =  frs$lat, lon =  frs$lon))
 # blockpoints may need to be downloaded using dataload_from_aws()
 table(latlon_is.valid(lat =  blockpoints$lat, lon =  blockpoints$lon))

## End(Not run)
```

`latlon_join_on_blockid` *latlon_join_on_blockid - get lat,lon of each block internal point via blockid get expanded version of sites2blocks data.table, with new lat,lon columns*

Description

latlon_join_on_blockid - get lat,lon of each block internal point via blockid get expanded version of sites2blocks data.table, with new lat,lon columns

Usage

```
latlon_join_on_blockid(s2b)
```

Arguments

`s2b`	like testoutput_getblocksnearby_10pts_1miles, output of `getblocksnearby()`

Value

returns the input data.table but with lat,lon columns added as block coordinates

Examples

```
s2b = copy(testoutput_getblocksnearby_10pts_1miles)
latlon_join_on_blockid(s2b) # done by trilaterate also
```

`lat_alias` *lat_alias, lon_alias (DATA) Synonyms for lat and lon*

Description

lists of synonyms for "latitude" and "longitude" used when guessing which column is what in user-provided tables of coordinates

`linesofcode2` *linesofcode2 - UTILITY - count lines of source code per .R file (not per function) - NOT TESTED*

Description

linesofcode2 - UTILITY - count lines of source code per .R file (not per function) - NOT TESTED

Usage

```
linesofcode2(
  folder = getwd(),
  packages,
  recursive = TRUE,
  sums = FALSE,
  rfolderonly = FALSE,
  cropfilename = 40,
  croppath = 20,
  showrows = NULL
)
```

Arguments

`folder`	path to folder with .R files
`packages`	optional vector of names of packages of source code
`recursive`	logical, look in subfolders
`sums`	logical, if TRUE, returns sums info, otherwise just prints that to console and returns more info
`rfolderonly`	logical
`cropfilename`	number of character to truncate filename to for display in console
`croppath`	limit path for display
`showrows`	optional

Value

data.frame of info about files

`lookup_pctile` *lookup_pctile - Find approx percentiles in lookup table that is in memory*

Description

lookup_pctile - Find approx percentiles in lookup table that is in memory

Usage

```
lookup_pctile(
  myvector,
  varname.in.lookup.table,
  lookup = usastats,
  zone = "USA"
)
```

Arguments

myvector	Numeric vector, required. Values to look for in the lookup table.
varname.in.lookup.table	Character element, required. Name of column in lookup table to look in to find interval where a given element of myvector values is. *** If vector is provided, then must be same length as myvector, but only 1 value for zone can be provided.
lookup	Either lookup must be provided, not quoted, or a lookup table called usastats must already be in memory. This is the lookup table data.frame with a PCTILE column, REGION column, and column whose name is the value of varname.in.lookup.table To use state lookups set lookup=statestats
zone	Character element (or vector as long as myvector), optional. If specified, must appear in a column called REGION within the lookup table, or NA returned for each item looked up and warning given. For example, it could be "NY" for New York State, "USA" for national percentiles.

See Also

Identical to pctile_from_raw_lookup() usastats statestats

---

mact_table *mact_table (DATA) MACT NESHAP subparts (the code and the description)*

Description

This is a data.table with one row per MACT subpart such as BBBBBB or ZZZZZZ and the title of each category. From the ECHO download of ICIS Air @seealso frs_by_mact

---

mapfastej_counties *mapfastej_counties - Static or HTML/leaflet map of counties*

Description

mapfastej_counties - Static or HTML/leaflet map of counties

Usage

```
mapfastej_counties(
  mydf,
  colorvarname = "pctile.Demog.Index.Supp",
  static_not_leaflet = FALSE,
  main = "Selected Counties",
  ...
)
```

Arguments

mydf	something like ejamit(fips = fips_counties_from_statename("Kentucky"), radius = 0)$results_bysite
colorvarname	colname of indicator in mydf that drives color-coding
static_not_leaflet	set TRUE to use map_shapes_plot() instead of map_shapes_leaflet()
main	title for map
...	passed to map_shapes_plot() if relevant

Value

leaflet html widget (but if static_not_leaflet=T, returns just shapes_counties_from_countyfips(mydf$siteid))

Examples

```
## Not run:
 fips_ky <- fips_counties_from_statename("Kentucky")
 x <- ejamit(fips = fips_ky, radius = 0)
 mapfastej_counties(x$results_bysite)

## End(Not run)
```

---

mapfast_gg *mapfast_gg - A very simple ggplot2 map of points in the USA*

Description

mapfast_gg - A very simple ggplot2 map of points in the USA

Usage

```
mapfast_gg(
  mydf = data.frame(lat = 40, lon = -100)[0, ],
  dotsize = 1,
  ptcolor = "black",
  xlab = "Longitude",
  ylab = "Latitude",
  ...
)
```

Arguments

mydf	data.frame with columns named lat and lon
dotsize	optional, size of dot representing a point
ptcolor	optional, color of dot
xlab	optional, text for x label
ylab	optional, text for y label
...	optional, passed to ggplot2::labs()

Value

a ggplot() object

Examples

```
## Not run:
  mapfast_gg(EJAM::testpoints_10)

  pts <- read.table(textConnection(
  "lat lon
  39.5624775 -119.7410994
  42.38748056 -94.61803333"
  ),
  header = TRUE,
  as.is = TRUE
  )
  mapfast_gg(pts)
  # str(pts) # lon, not long

## End(Not run)
```

---

map_blockgroups_over_blocks

map_blockgroups_over_blocks - Overlay blockgroups near 1 site, after plotblocksnearby() Overlay blockgroups near 1 site, after plotblocksnearby()

Description

map_blockgroups_over_blocks - Overlay blockgroups near 1 site, after plotblocksnearby() Overlay blockgroups near 1 site, after plotblocksnearby()

Usage

```
map_blockgroups_over_blocks(y)
```

Arguments

y	output of plotblocksnearby()

Value

leaflet map widget

See Also

map_blockgroups()

Examples

```
dontrun{
 y <- plotblocksnearby(testpoints_10[5,],
        radius = 3,
        returnmap = TRUE)
 map_blockgroups_over_blocks(y)
  }
```

---

map_facilities *map_facilities*

Description

make a leaflet map of uploaded points

Usage

```
map_facilities(mypoints, rad = 3, highlight = FALSE, clustered)
```

Arguments

mypoints,	data frame of uploaded points
rad,	a size for drawing each circle (buffer search radius)
highlight,	a logicial for whether to highlight overlapping points (defaults to FALSE)
clustered,	a vector of T/F values for each point, indicating if they overlap with another

Value

a leaflet map with circles, circleMarkers, and basic popup

map_facilities_proxy *map_facilities_proxy*

Description

update a leaflet map within the EJAM shiny app with uploaded points.

Usage

```
map_facilities_proxy(
  mymap,
  rad = 3,
  highlight = FALSE,
  clustered = FALSE,
  popup_vec = NULL,
  use_marker_clusters = FALSE
)
```

Arguments

mymap,	leafletProxy map object to be added to
rad,	a size for drawing each circle (buffer search radius)
highlight,	a logicial for whether to highlight overlapping points (defaults to FALSE)
clustered,	a vector of T/F values for each point, indicating if they overlap with another
popup_vec,	a vector of popup values to display when points are clicked. Length should match number of rows in the dataset.
use_marker_clusters,	boolean for whether to group points into markerClusters. Uses logic from shiny app to only implement when n > 1000.

Value

a leaflet map with circles, circleMarkers, and basic popup

map_shapes_leaflet *map_shapes_leaflet*

Description

map_shapes_leaflet

Usage

```
map_shapes_leaflet(shapes, color = "green", popup = shapes$NAME)
```

Arguments

shapes	like from shapes_counties_from_countyfips()
color	passed to leaflet::addPolygons()
popup	passed to leaflet::addPolygons()

Value

html widget from leaflet::leaflet()

`map_shapes_mapview` *map_shapes_mapview*

Description

map_shapes_mapview

Usage

```
map_shapes_mapview(shapes, col.regions = "green", map.types = "OpenStreetMap")
```

Arguments

`shapes`	like from shapes_counties_from_countyfips()
`col.regions`	passed to mapview() from mapview package
`map.types`	passed to mapview() from mapview package

`map_shapes_plot` *map_shapes_plot*

Description

map_shapes_plot

Usage

```
map_shapes_plot(shapes, main = "Selected Census Units", ...)
```

Arguments

`shapes`	like from shapes_counties_from_countyfips()
`main`	title for map
`...`	passed to plot()

Value

Just draws map using plot()

`metadata_add` *helper function for package to set attributes of a dataset*

Description

This can be used annually to update some datasets in a package. It just makes it easier to set a few metadata attributes similarly for a number of data elements, for example, to add new or update existing attributes.

Usage

```
metadata_add(x, metadata)
```

Arguments

`x`	dataset (or any object) whose metadata you want to update or create
`metadata`	must be a named list, so that the function can do this for each i: `attr(x, which=names(metadata)[i]) <- metadata[[i]]`

Value

returns x but with new or altered attributes

See Also

metadata_check()

Examples

```
x <- data.frame(a=1:10,b=1001:1010)
metadata <- list(
ejscreen_version =  '2.2',
acs_version =          '2017-2021',
census_version = 2020,
ejscreen_releasedate = '2023-06-23',
acs_releasedate =      '2022-12-08',
ejscreen_pkg_data = NA
)
x <- metadata_add(x, metadata)
attributes(x)
x <- metadata_add(x, list(status='final'))
attr(x,'status')
```

metadata_check *helper function in updating the package metadata*

Description

Quick and dirty helper during development, to check all the attributes of all the data files in relevant packages. It loads unloaded packages as needed, which you might not want it to do, but it is not coded to be able to check attributes without doing that.

Usage

```
metadata_check(
  packages = EJAM::ejampackages,
  which = c("census_version", "acs_version", "acs_releasedate", "ACS",
   "ejscreen_version", "ejscreen_releasedate", "ejscreen_pkg_data", "year", "released"),
  loadifnotloaded = TRUE
)
```

Arguments

packages	Optional. e.g. 'EJAMejscreendata', or can be a vector of character strings, and if not specified, default is to report on all packages with EJ as part of their name, like EJAMejscreenapi
which	Optional vector (not list) of strings, the attributes. Default is some typical ones used in EJAM-related packages currently.
loadifnotloaded	Optional to control if func should temporarily attach packages not already loaded.

NAICS *NAICS (DATA) named list of all NAICS code numbers and industry name for each*

Description

NAICS (DATA) named list of all NAICS code numbers and industry name for each

Details

see https://naics.com

See Also

naicstable naics_from_any() naics_categories() NAICS

naics2children — *naics2children - See NAICS codes queried plus all children of any of those Used by naics_find()*

Description

naics2children - See NAICS codes queried plus all children of any of those Used by naics_find()

Usage

```
naics2children(codes, allcodes = EJAM::NAICS)
```

Arguments

codes	vector of numerical or character
allcodes	Optional (already loaded with package) - dataset with all the codes

Details

start with shortest (highest level) codes. since tied for nchar, these branches have zero overlap, so do each. for each of those, get its children = all rows where parentcode == substr(allcodes, 1, nchar(parentcode)) put together list of all codes we want to include so far. now for the next longest set of codes in original list of codes, do same thing. etc. until did it for 5 digit ones to get 6digit children. take the unique(allthat) table(nchar(as.character(NAICS))) 2 3 4 5 6 17 99 311 709 1057

Value

vector of codes and their names

See Also

naics_find() NAICS

Examples

```
naics2children(211)
naics_find(211, exactnumber=TRUE)
naics_find(211, exactnumber=TRUE, add_children = TRUE)
NAICS[211][1:3] # wrong
NAICS[NAICS == 211]
NAICS["211 - Oil and Gas Extraction"]
```

`naicstable` *naicstable (DATA) data.table of NAICS code(s) and industry names for each EPA-regulated site in Facility Registry Service Also has the 2,3,4,5,and 6-digit NAICS that this code falls under, where relevant for given length*

Description

This is similar to the data file EJAM::NAICS but in a more useful format and newer functions work with it.

Details

see https://naics.com

See Also

`naics_from_any()` NAICS `naics_categories()` `naics_findwebscrape()`

`naics_categories` *naics_categories - See the names of industrial categories and their NAICS code Easy way to list the 2-digit NAICS (17 categories), or other level*

Description

naics_categories - See the names of industrial categories and their NAICS code Easy way to list the 2-digit NAICS (17 categories), or other level

Usage

```
naics_categories(digits = 2, dataset = EJAM::NAICS)
```

Arguments

`digits`	default is 2, for 2-digits NAICS, the top level, but could be up to 6.
`dataset`	Should default to the dataset called NAICS, installed with this package. see NAICS Check attr(NAICS, 'year')

Details

Also see https://www.naics.com/search/

There are this many NAICS codes roughly by number of digits in the code:

table(nchar(NAICS))

2 3 4 5 6

17 99 311 709 1057

See https://www.census.gov/naics/

See Also

naics_from_any NAICS

Examples

```
naics_categories()
```

`naics_download` *naics_download - script to download NAICS file with code and name of sector*

Description

See source code. Mostly just a short script to get the 2017 or 2022 codes and names. See <'https://www.census.gov/naics/

Usage

```
naics_download(
  year = 2017,
 urlpattern = "https://www.census.gov/naics/YYYYNAICS/2-6%20digit_YYYY_Codes.xlsx",
  destfile = paste0("~/Downloads/", year, "NAICS.xlsx")
)
```

Arguments

`year`	which vintage of NAICS codes to use, 2012, 2017, or 2022
`urlpattern`	full url of xlsx file to use, but with YYYY instead of year
`destfile`	full path and name of file to save as locally

Value

names list with year as an attribute

`naics_findwebscrape` *naics_findwebscrape - for query term, show list of roughly matching NAICS, scraped from web This finds more than just `naics_from_any()` does, since that needs an exact match but this looks at naics.com website which lists various aliases for a sector.*

Description

naics_findwebscrape - for query term, show list of roughly matching NAICS, scraped from web This finds more than just `naics_from_any()` does, since that needs an exact match but this looks at naics.com website which lists various aliases for a sector.

Usage

```
naics_findwebscrape(query)
```

Arguments

query	text like "gasoline" or "copper smelting"

Value

data.frame of info on what was found, naics and title

See Also

naics_from_any() url_naics.com()

Examples

```
 # naics_from_any("copper smelting")
 # naics_from_any("copper smelting", website_scrape=TRUE)
 # browseURL(naics_from_any("copper smelting", website_url=TRUE) )

  url_naics.com("copper smelting")
  ## Not run:
  naics_findwebscrape("copper smelting")
  browseURL(url_naics.com("copper smelting"))
  browseURL(naics_url_of_code(326))

## End(Not run)
```

---

naics_from_any — *naics_from_any - General way to search for industry names and NAICS codes Find industry names and codes by searching for queried code(s) or text*

Description

naics_from_any - General way to search for industry names and NAICS codes Find industry names and codes by searching for queried code(s) or text

Usage

```
naics_from_any(
  query,
  children = FALSE,
  ignore.case = TRUE,
  fixed = FALSE,
  website_scrape = FALSE,
  website_url = FALSE
)
```

Arguments

query	query string(s) and/or number(s), vector of NAICS codes or industry names or any regular expression or partial words
children	logical, if TRUE, also return all the subcategories - where NAICS starts with the same digits

ignore.case	see grepl()
fixed	should it be an exact match? see grepl()
website_scrape	whether to scrape info from the NAICS website to return a table of codes and names that match (web query uses synonyms so gets more hits)
website_url	whether to return the URL of the webpage with info on the NAICS (web query uses synonyms so gets more hits)

Value

a subset of the naicstable data.table (not just the codes column)

See Also

naics_subcodes_from_code() naics_from_code() naics_from_name() naics_from_any()

Examples

```
# Also see vignette for examples
  naics_categories()
  naics_from_any(naics_categories(3))[order(name),.(name,code)][1:10,]
  naics_from_any(naics_categories(3))[order(code),.(code,name)][1:10,]
  naics_from_code(211)
  naicstable[code==211,]
  naics_subcodes_from_code(211)
  naics_from_code(211,  children = TRUE)
  naicstable[n3==211,]
  NAICS[211][1:3] # wrong
  NAICS[NAICS == 211]
  NAICS["211 - Oil and Gas Extraction"]

 naics_from_any("plastics and rubber")[,.(name,code)]
 naics_from_any(326)
 naics_from_any(326, children = T)[,.(code,name)]
 naics_from_any("plastics", children=T)[,unique(n3)]
 naics_from_any("pig")
 naics_from_any("pig ") # space after g

 # naics_from_any("copper smelting")
 # naics_from_any("copper smelting", website_scrape=TRUE)
 # browseURL(naics_from_any("copper smelting", website_url=TRUE) )

 a = naics_from_any("plastics")
 b = naics_from_any("rubber")
 fintersect(a,b)[,.(name,code)] #  a AND b
 funion(a,b)[,.(name,code)]     #  a OR  b
 naics_subcodes_from_code(funion(a,b)[,code])[,.(name,code)]   #  plus children
 naics_from_any(funion(a,b)[,code], children=T)[,.(name,code)] #  same

 NROW(naics_from_any(325))
#[1] 1
 NROW(naics_from_any(325, children = T))
#[1] 54
 NROW(naics_from_any("chem"))
#[1] 20
 NROW(naics_from_any("chem", children = T))
```

```
[1] 104
```

`naics_from_code` *naics_from_code - search for industry names by NAICS code(s), 2-6 digits long each See* `naics_from_any()` *which uses this*

Description

naics_from_code - search for industry names by NAICS code(s), 2-6 digits long each See `naics_from_any()` which uses this

Usage

```
naics_from_code(mycodes, children = FALSE)
```

Arguments

`mycodes`	vector of numeric NAICS codes. see `https://naics.com`
`children`	logical, if TRUE, also return all the subcategories - where NAICS starts with the same digits

Value

a subset of the naicstable data.table (not just the codes column)

See Also

`naics_subcodes_from_code()` `naics_from_code()` `naics_from_name()` `naics_from_any()`

`naics_from_federalregister` *naics_from_federalregister - DRAFT WORK IN PROGRESS*

Description

naics_from_federalregister - DRAFT WORK IN PROGRESS

Usage

```
naics_from_federalregister(naics_text_copy_from_fr)
```

Arguments

`naics_text_copy_from_fr`

naics_from_name *naics_from_name - search for industry names and NAICS codes by query string query by parts of words, etc. in the industry name. See* naics_from_any() *which uses this*

Description

naics_from_name - search for industry names and NAICS codes by query string query by parts of words, etc. in the industry name. See naics_from_any() which uses this

Usage

```
naics_from_name(mynames, children = FALSE, ignore.case = TRUE, fixed = FALSE)
```

Arguments

mynames	query string, vector of NAICS industry names or any regular expression or partial words. See https://naics.com
children	logical, if TRUE, also return all the subcategories - where NAICS starts with the same digits
ignore.case	see grepl()
fixed	should it be an exact match? see grepl()
search_on_naics_website	whether to query on naics website for more hits than just search for text in industry title

Value

a subset of the naicstable data.table (not just the codes column)

See Also

naics_subcodes_from_code() naics_from_code() naics_from_name() naics_from_any()

Examples

```
data.table::fintersect(naics_from_any( "manufac"), naics_from_any("chem"))
```

naics_subcodes_from_code *naics_subcodes_from_code - find subcategories of the given overall NAICS industry code(s) Given 3-digit NAICS code, for example, get all NAICS that start with those digits.*

Description

naics_subcodes_from_code - find subcategories of the given overall NAICS industry code(s) Given 3-digit NAICS code, for example, get all NAICS that start with those digits.

Usage

```
naics_subcodes_from_code(mycodes)
```

Arguments

mycodes	NAICS codes vector, of 2 to 6 digits each. See https://naics.com

Details

similar idea was naics2children() but this is more robust See naics_from_any() which uses this

Value

a subset of the naicstable data.table (not just the codes column)

See Also

naics_subcodes_from_code() naics_from_code() naics_from_name() naics_from_any()

Examples

```
naics_categories()
```

---

naics_url_of_code *naics_url_of_code - Get URL for page with info about industry sector(s) by NAICS See (https://naics.com) for more information on NAICS codes*

Description

naics_url_of_code - Get URL for page with info about industry sector(s) by NAICS See (https://naics.com) for more information on NAICS codes

Usage

```
naics_url_of_code(naics)
```

Arguments

naics	vector of one or more NAICS codes, like 11,"31-33",325

Value

vector of URLs as strings like https://www.naics.com/six-digit-naics/?v=2017&code=22

naics_validation *naics_validation - Validate NAICS uploads*

Description

Validates and prepares echo uploads

Usage

```
naics_validation(naics_enter, naics_select)
```

Arguments

naics_enter	vector of naics
naics_select	

Value

boolean value (valid or not valid) - TRUE if length of at least one of the two input vectors is > 0

names_d *names_d (DATA) list of demographic indicator names*

Description

names_d (DATA) list of demographic indicator names

See Also

map_headernames (in EJAMejscreenapi package) names_d names_e namez

names_e *names_e (DATA) list of environmental indicator names*

Description

names_e (DATA) list of environmental indicator names

See Also

EJAMejscreenapi dataset called map_headernames names_d names_e namez

names_whichlist *names_whichlist See which of the lists of names a single term appears in*

Description

names_whichlist See which of the lists of names a single term appears in

Usage

```
names_whichlist(
  x,
  exact = T,
  grepmatching = T,
  ignore.case.exact = FALSE,
  ignore.case.grep = FALSE,
  keylists = F,
  exactonly = FALSE
)
```

Arguments

x	term, like part or all of a variable name, such as state.avg
exact	whether to look for exact matches
grepmatching	whether to look for matches via grep (partial match)
ignore.case.exact	whether to ignore capitalization in exact matches
ignore.case.grep	passed to grep as ignore.case param
keylists	if true, only report for the key lists not friendly, all, these, need types.
exactonly	to limit output to rows with exact matches

Details

EJAM::namez has a list of lists of names used for indicators or variables, such as namez$d_friendly which is a vector of terms like "Demog.Ind.", "Suppl Demog Index", "% Low-inc.", etc.

Value

a data.frame of whichlist, exactmatch, grepmatch, and grephits (examples)

Examples

```
x <- names_whichlist("rsei", ignore.case.exact = T, ignore.case.grep = T)
subset(x, !grepl("friendly", x$whichlist))

subset(x,  grepl("friendly", x$whichlist))
subset(namez, names(namez) != "all_r" & names(namez) %in%
   subset(x, x$grepmatch == "yes" & !grepl("friendly", x$whichlist))$whichlist  )
grep("\\.eo$", namez$ej, value = T)
```

names_whichlist_multi *names_whichlist_multi See which lists of names the given indicator names are in*

Description

names_whichlist_multi See which lists of names the given indicator names are in

Usage

```
names_whichlist_multi(x, ...)
```

Arguments

x	vector of names (query terms)
...	passed to names_whichlist()

Value

a list of sets of names

names_whichlist_multi_key *names_whichlist_multi_key See which key lists of names the given indicator names are in*

Description

names_whichlist_multi_key See which key lists of names the given indicator names are in

Usage

```
names_whichlist_multi_key(x, ...)
```

Arguments

x	vector of names
...	passed to names_whichlist_multi()

Value

vector maybe

namez *namez (DATA) list of lists of indicator names (complete list in 1 object)*

Description

namez (DATA) list of lists of indicator names (complete list in 1 object)

See Also

EJAMejscreenapi dataset called map_headernames names_d names_e

pctiles_lookup_create *pctiles_lookup_create - create lookup table of percentiles 0 to 100 and mean for each indicator by State or USA total*

Description

pctiles_lookup_create - create lookup table of percentiles 0 to 100 and mean for each indicator by State or USA total

Usage

```
pctiles_lookup_create(
  x,
  zone.vector = NULL,
  zoneOverallName = "USA",
  wts = NULL,
  usecollapse = TRUE,
  type = 7
)
```

Arguments

x	data.frame with numeric data. Each column will be examined to calculate mean, and percentiles, for each zone
zone.vector	optional names of states or regions, for example. same length as wts, or rows in mydf
zoneOverallName	optional. Default is USA.
wts	leave as default since weighted percentiles of blockgroups are not used for EJScreen percentiles anymore
usecollapse	logical, whether to use collapse::fquantile() instead of Hmisc package wtd.quantile and stats pkg quantile, to test before fully removing dependency on Hmisc and also speed it up.

type — DO NOT CHANGE - moot for EJScreen/EJAM - SEE SOURCE CODE - Hmisc pkg wtd.quantile type "1/n" was used here in the past and possibly by EJScreen (EJScreen no longer uses weighted percentiles so this is moot for the weighted case) but collapse pkg fquantile is now used here to avoid Hmisc dependency and fquantile type 4 seems to be the same as Hmisc type "1/n" but that has not been confirmed, and this function by default uses fquantile type 1, the inverse of the ECDF however, which seems simpler than using type 4 which does linear interpolation between points of the ECDF! *** NEED TO CONFIRM IF THAT CREATES A TABLE DIFFERENT THAN WHAT EJSCREEN WOULD CREATE

Details

EJScreen assigns each indicator in each block group a percentile value via python script, using https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.percentileofscore.html

The way the python function is used as of 2023 is that percentileofscore is 80% if

80% of all indicator values (statewide or nationwide, depending on the type being calculated)

are less than (NOT equal to) the indicator value

in the specified block group (since kind="strict").

The percentile recorded in the EJScreen dataset is the floor of that,

meaning if the 81.9% of values are less than x, the percentile is reported as 81.

The EJScreen python script used to create percentile lookup tables is in a file

called cal_statepctile_0222.py and the key lines of code and functions it uses are

pctile = math.floor(stats.percentileofscore(barray, indicatorscore, kind="strict"))

binvalue = getBinvalue(pctile)

and

def getBinvalue(pct):

if pct is None: return 0 else: if pct >= 95: return 11 elif pct >= 90 and pct < 95: return 10 elif pct >= 80 and pct < 90: return 9 elif pct >= 70 and pct < 80: return 8 elif pct >= 60 and pct < 70: return 7 elif pct >= 50 and pct < 60: return 6 elif pct >= 40 and pct < 50: return 5 elif pct >= 30 and pct < 40: return 4 elif pct >= 20 and pct < 30: return 3 elif pct >= 10 and pct < 20: return 2 else: return 1

---

pctile_from_raw_lookup

pctile_from_raw_lookup - Find approx percentiles in lookup table that is in memory

---

Description

This is used with a lookup table to convert a raw indicator vector to percentiles in US or States.

Usage

```
pctile_from_raw_lookup(
  myvector,
  varname.in.lookup.table,
  lookup = usastats,
  zone = "USA"
)
```

Arguments

myvector	Numeric vector, required. Values to look for in the lookup table.
varname.in.lookup.table	Character element, required. Name of column in lookup table to look in to find interval where a given element of myvector values is. *** If vector is provided, then must be same length as myvector, but only 1 value for zone can be provided.
lookup	Either lookup must be provided, not quoted, or a lookup table called usastats must already be in memory. This is the lookup table data.frame with a PCTILE column, REGION column, and column whose name is the value of varname.in.lookup.table To use state lookups set lookup=statestats
zone	Character element (or vector as long as myvector), optional. If specified, must appear in a column called REGION within the lookup table, or NA returned for each item looked up and warning given. For example, it could be "NY" for New York State, "USA" for national percentiles.

Details

This could be recoded to be more efficient - could use data.table.

The data.frame lookup table must have a field called "PCTILE" that has quantiles/percentiles and other column(s) with values that fall at those percentiles. usastats and statestats are such lookup tables. This function accepts lookup table (or uses one called us if that is in memory), and finds the number in the PCTILE column that corresponds to where a specified value (in myvector) appears in the column called varname.in.lookup.table. The function just looks for where the specified value fits between values in the lookup table and returns the approximate percentile as found in the PCTILE column. If the value is between the cutpoints listed as percentiles 89 and 90, it returns 89, for example. If the value is exactly equal to the cutpoint listed as percentile 90, it returns percentile 90. If the value is exactly the same as the minimum in the lookup table and multiple percentiles in that lookup are listed as tied for having the same threshold value defining the percentile (i.e., a large percent of places have the same score and it is the minimum score), then the percentile gets reported as 0, not the percent of places tied for that minimum score. Note this is true whether they are tied at a value of 0 or are tied at some other minimum value than 0. If the value is less than the cutpoint listed as percentile 0, which should be the minimum value in the dataset, it still returns 0 as the percentile, but with a warning that the value checked was less than the minimum in the dataset.

It also handles other odd cases, like where a large percent of all raw scores are tied at the minimum value, in which case it reports 0 as percentile, not that large percent.

Value

By default, returns numeric vector length of myvector.

Examples

```
## Not run:

eg <- dput(round(as.vector(unlist(testoutput_ejamit_10pts_1miles$results_overall[ , ..names_d] )),3))

data.frame(value = eg, pctile = t(testoutput_ejamit_10pts_1miles$results_overall[ , ..names_d_pctile]))

data.frame(value = eg, pctile = lookup_pctile(eg, names_d))


  # compare ejscreen API output percentiles to those from this function:
  for (vname in c(names_d[c(1,3:6,8:10)] )) {
     print(pctile_from_raw_lookup(testoutput_ejscreenapi_plus_100[,vname] / 100, vname,
       lookup = usastats)
       - testoutput_ejscreenapi_plus_100[,paste0("pctile.",vname)] )
  }
  for (vname in c(names_e )) {
     print(pctile_from_raw_lookup(testoutput_ejscreenapi_plus_100[,vname], vname,
       lookup = usastats)
         - testoutput_ejscreenapi_plus_100[,paste0("pctile.",vname)] )
  }

## End(Not run)
```

`plotblocksnearby` *plotblocksnearby - Map view of Census blocks (their centroids) near one or more sites Utility to quickly view one or more facility points on map with the blocks found nearby*

Description

plotblocksnearby - Map view of Census blocks (their centroids) near one or more sites Utility to quickly view one or more facility points on map with the blocks found nearby

Usage

```
plotblocksnearby(
  sitepoints,
  radius = 3,
  sites2blocks,
  siteidvarname = "ejam_uniq_id",
  usemapfast = TRUE,
  returnmap = FALSE,
  maxradius = 31.07,
  avoidorphans = FALSE,
  ...
)
```

Arguments

`sitepoints`	table of points with lat, lon in decimal degrees (data.frame or data.table), but also could just be the output of getblocksnearby() if that has already been done.

radius	optional. in miles (Max allowed is 32 miles, or just over 50 kilometers since 31.06856 miles is 50 * 1000 / meters_per_mile).
sites2blocks	optional. If provided, used as sites2blocks like testoutput_getblocksnearby_10pts_1miles If neither sites2blocks nor sitepoints is provided it cannot plot and returns error. If sites2blocks and sitepoints are both provided, it uses them both to plot blocks and sites (centers of circles). If sites2blocks not provided, but sitepoints alone is provided, checks if sitepoints is actually sites2blocks, and uses as such. If sites2blocks not provided, but sitepoints alone is provided, and sitepoints is really sitepoints, it runs getblocksnearby() to create sites2blocks. If sites2blocks is provided, but sitepoints is not, it could only do a bad approximation of sitepoints so it will not draw the circle or site at center of the circle.
siteidvarname	optional. specifies the column name in sites2blocks that is the unique site id, the values of which should also be the row numbers of the corresponding sites in sitepoints, with a site appearing once in sitepoints, and in sites2blocks appearing once per block that is near that site.
usemapfast	optional. simpler plot if FALSE
returnmap	optional. if set TRUE, returns the leaflet map object instead of tabular info.
maxradius	optional. see getblocksnearby()
avoidorphans	optional. see getblocksnearby()
...	optional. passed to mapfast() or plot() depending on usemapfast

Details

Uses getblocksnearby() if lat,lon points provided as sitepoints, but skips it if looks like user passed output of getblocksnearby(), and then displays a map of those blocks near the specified point.

Value

invisibly returns sites2blocks like getblocksnearby() does

Examples

```
#  see all Census Blocks within 1 mile of 1 site, if already had run getblocksnearby()
getblocks_output <- copy(testoutput_getblocksnearby_10pts_1miles)
if ("siteid" %in% names(getblocks_output)) {
siteidvarname <- "siteid" # the old default
eg <- getblocks_output[siteid == 1,]
eg2 <- getblocks_output[siteid %in% c(4,10),]
} else {
siteidvarname <- "ejam_uniq_id"
 eg <- getblocks_output[ejam_uniq_id == 1,]
 eg2 <- getblocks_output[ejam_uniq_id %in% c(4,10),]
}
 z <-  plotblocksnearby(sitepoints = testpoints_10,
   sites2blocks = eg, radius = 1)
 # see two sites if already had run getblocksnearby()
 z2 <-  plotblocksnearby(sitepoints = testpoints_10[c(4,10),],
   sites2blocks = eg2, radius = 1)
## Not run:
# See one randomly selected regulated facility from FRS and all Census Blocks within 2 miles:
  plotblocksnearby(testpoints_n(1), 2)
# See two sites and all Census Blocks within 5 kilometers
```

```
 plotblocksnearby(testpoints_2, radius = convert_units(5, from = "km", towhat = "miles"))
# See 100 sites and all blocks within 1 mile of each -
# Note you have to specify radius here or it uses default that may not match intent
# - and this is a bit slow
plotblocksnearby(testoutput_ejamit_100pts_1miles$results_bysite[, c(siteidvarname, "lat", "lon"), with=FALS
   radius = 1)

## End(Not run)
```

`plot_barplot_ratios` *plot_barplot_ratios Make barplot of ratios of demographic score to its average*

Description

plot_barplot_ratios Make barplot of ratios of demographic score to its average

Usage

```
plot_barplot_ratios(
  ratio.to.us.d.overall,
  names2plot_friendly = NULL,
  mycolorsavailable = c("gray", "yellow", "orange", "red")
)
```

Arguments

`ratio.to.us.d.overall`
: named list of a few ratios to plot

`names2plot_friendly`
: names to use for plot - should be same length as named list ratio.to.us.d.overall

`mycolorsavailable`
: leave as default

Details

For plots in general, see:

- https://echarts4r.john-coene.com/articles/themes.html
- https://exts.ggplot2.tidyverse.org/gallery

For BARPLOTS, see/ merge/consolidate:

- output$view1_summary_plot <- renderPlot(v1_summary_plot()) and v1_summary_plot <- reactive( ) in EJAM server for Short Report if bar type
- output$summ_display_bar <- renderPlot( ) contains its own plot code not a reactive in EJAM server for tab showing barplots in Detailed Results
- plot_barplot_ratios() drafted function in EJAM

For BOXPLOTS, see:

- v1_summary_plot <- reactive( ) and output$view1_summary_plot <- renderPlot(v1_summary_plot())
  in EJAM server for SHORT report if box type, and
  in EJAM server for LONG report passed as a parameter
- boxplots_ratios() in EJAMejscreenapi
  (NOT in EJAM server for Detailed Results interactive views)
- ejscreenapi_script() code also relevant? in EJAMejscreenapi
- box/scatter examples in ggplot, https://r-graph-gallery.com/89-box-and-scatter-plot-with-ggplot2.html
- boxplots in base R, https://www.r-bloggers.com/2023/09/how-to-reorder-boxplots-in-r-a-comprehen

For HISTOGRAMS, see:

- output$summ_display_hist <- renderPlot in EJAM server for interactive views
- the histograms code and discussion in EJAMbatch.summarizer package

Value

ggplot should be returned

See Also

table_xls_format() plot_barplot_ratios

Examples

```
plot_barplot_ratios(unlist(testoutput_ejamit_1000pts_1miles$results_overall[ , c(..names_d_ratio_to_avg , 
```

---

plot_demogshare_by_distance

plot_demogshare_by_distance - work in progress

Description

plot_demogshare_by_distance - work in progress

Usage

```
plot_demogshare_by_distance(
  results_bybg_people,
  demogvarname = names_d[1],
  siteids = unique(results_bybg_people$siteid),
  show.lowess = F,
  show.lm = TRUE,
  show.line = TRUE,
  ...
)
```

Arguments

results_bybg_people	
demogvarname	
siteids	
show.lowess	F
show.lm	linefit
show.line	linefit
...	passed to plot

Details

Could also consider plotting something like boxplot(demogvar ~ round(distance, 1))

See notes on plots at plot_barplot_ratios()

---

plot_distance_by_pctd *plot_distance_by_pctd - Plot percent demographics within X miles of a site*

Description

plot_distance_by_pctd - Plot percent demographics within X miles of a site

Usage

```
plot_distance_by_pctd(
  s2b = NULL,
  mysiteid = NULL,
  myvars = c(names_d_count, names_d_subgroups_count)[1],
  dpctvar = paste0("pct", myvars)
)
```

Arguments

s2b	output of getblocksnearby()
mysiteid	one number that is the siteid to look at in s2b
myvars	a colname of a population count variable in blockgroupstats indicating which to plot, like "hisp" or "lowinc" and only works for one indicator at a time so far.
dpctvar	a colname of usastats and statestats that is the percentage version of myvars, like "pcthisp" or "pctlowinc"

Value

returns s2b but with more columns in it, like cumpop, cumdpop, pctdwithin

Examples

```
plot_distance_by_pctd()
```

plot_distance_cdf_by_group *plot_distance_cdf_by_group - SLOW - needs to be optimized CDF Line Plots of cumulative share of each demographic group, within each distance Each groups distribution of distances*

Description

plot_distance_cdf_by_group - SLOW - needs to be optimized CDF Line Plots of cumulative share of each demographic group, within each distance Each groups distribution of distances

Usage

```
plot_distance_cdf_by_group(
  results_bybg_people = NULL,
  radius_miles =
   round(max(results_bybg_people$distance_min_avgperson[!is.infinite(results_bybg_people$distanc
    na.rm = T), 1),
  subgroups_type = NULL,
  demogvarname = NULL,
  demoglabel = NULL,
  colorlist = colorspace::diverging_hcl(length(demogvarname)),
  coloroverall = "black",
  returnwhat = "table",
  ...
)
```

Arguments

results_bybg_people	data.table from doaggregate()$results_bybg_people
radius_miles	miles radius that was max distance analyzed
subgroups_type	optional, can be set to "nh" or "alone". Specifies types of race ethnicity subgroups to use for demogvarname but only if demogvarname is not specified as a parameter. If neither is specified it tries to use default_subgroups_type if that is a variable set by global.R, since it cannot check the reactive variable input$subgroups_type outside the context of the web app.
demogvarname	optional way to specify names of columns to use from results_bybg_people, e.g., c("pctlowinc", "pctmin"), or namez$d, or could be a vector of subgroups such as namez$d_subgroups_nh that includes "pctnhba" etc. or namez$d_subgroups_alone that includes "pctba" etc., but if demogvarname is not specified here as a parameter, this info could also be specified by the subgroups_type parameter here. If neither is specified, the function will try to use a default (which may not reflect any changes being made during development of EJAM if default_subgroups_type is in flux)
demoglabel	friendly text names for labelling graphic, like "Low income residents"
colorlist	colors like "red" etc. for the demographic groups of interest
coloroverall	color like "gray" for everyone as a whole

returnwhat — If returnwhat is "table", invisibly returns a full table of sorted distances of block-groups, cumulative count of demog groups at that block group's distance. If returnwhat is "plotfilename" then it returns the full path including filename of a .png in a tempdir If returnwhat is "plot" then it returns the plot object as needed for table_xls_format()

Value

see returnwhat parameter

See Also

distance_by_group() ejamit() for examples

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

---

plot_distance_mean_by_group

plot_distance_mean_by_group - Barplot Avg. proximity, by group Shows proximity to sites, for each demographic group (relative to everyone else)

Description

plot_distance_mean_by_group - Barplot Avg. proximity, by group Shows proximity to sites, for each demographic group (relative to everyone else)

Usage

```
plot_distance_mean_by_group(
  results_bybg_people,
  demogvarname = NULL,
  demoglabel = NULL,
  graph = TRUE,
  returnwhat = "table"
)
```

Arguments

results_bybg_people	data.table from doaggregate()$results_bybg_people
demogvarname	vector of column names like "pctlowinc" etc.
demoglabel	vector of labels like "Low Income Residents" etc.
returnwhat	If returnwhat is "table", invisibly returns a data.frame with group, ratio, avg_distance_for_group, avg_distance_for_nongroup. If returnwhat is "plotfilename" then it returns the full path including filename of a .png in a tempdir If returnwhat is "plot" then it returns the plot object as needed for table_xls_format() ?

Details

Note that the ratio shown is a ratio of distance among others to distance of a given group, so values below 1 mean the given demographic group lives closer to facilities. A value of 0.85 would mean the group is only 85% as far from a site as everyone else.

Note it is in miles assuming input was in miles, and the distance for each resident is actually the average distance of all residents within their Census block (not block group), and when a site is very close to the block internal point (like a centroid) relative to the size of the block, the distance to the average resident in the block is estimated as 90 percent of the effective radius, which is what the radius of the block would be if it were the same area in square meters or miles but circular in shape. This is the approach used in EJScreen to estimate average proximity of a block resident in cases where the block is extremely close to the site or the site may actually be inside the block, or exactly on top of the internal point of the block, in which case zero would not be an appropriate estimate of the distance, hence this adjustment is made in EJAM getblocksnearby()

Value

see parameter returnwhat

See Also

distance_by_group()

distance_by_group_plot() plot_distance_cdf_by_group()

Examples

```
y <- ejamit(testpoints_100, radius = 3)
plot_distance_mean_by_group(y$results_bybg_people) # or distance_mean_by_group() synonym
print(distance_by_group(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income'))
distance_by_group_plot(y$results_bybg_people,
  demogvarname = 'pctlowinc', demoglabel = 'Low Income')
xyz = plot_distance_cdf_by_group(y$results_bybg_people) #
tail(round(xyz,3))
tail(xyz[xyz$pctwa <= 0.501, ]) #  Median distance to nearest site here
  for White Alone is 2.15 miles, but >60% of Black Alone have a site that close.
tail(xyz[xyz$pctba <= 0.501, ]) #  Median distance to nearest site here
  for Black Alone is 1.85 miles
round(tail(xyz[xyz$dist <=1, ]), 3) #  11% of White have a site within 1 mile,
  compared to 18.7% of Asian who do.
```

popshare_at_top_n *popshare_at_top_n - top N sites account for what percent of residents? What fraction of total population is accounted for by the top N places?*

Description

popshare_at_top_n - top N sites account for what percent of residents? What fraction of total population is accounted for by the top N places?

Usage

```
popshare_at_top_n(pop, n = 10, astext = FALSE, dig = 0)
```

Arguments

pop	vector of population totals across places, like out$results_bysite$pop where out is the output of ejamit()
n	the number of places to consider
astext	if TRUE, return text of description of results
dig	rounding digits for text output

Value

A fraction of 1

Examples

```
x <- testdata_ejamit_output_100pts_1miles$results_bysite
popshare_p_lives_at_what_pct(x$pop, p = 0.50, astext=TRUE)
popshare_p_lives_at_what_n(  x$pop, p = c(0.50, 0.67, 0.80, 0.95))
popshare_at_top_x_pct(       x$pop, x = c(0.25, 0.50, .90))
popshare_at_top_n(           x$pop, n = c(1, 5, 10))
```

popshare_at_top_x_pct *popshare_at_top_x_pct - top X percent of sites account for what percent of residents? What fraction of total population is accounted for by the top X percent of places?*

Description

popshare_at_top_x_pct - top X percent of sites account for what percent of residents? What fraction of total population is accounted for by the top X percent of places?

Usage

```
popshare_at_top_x_pct(pop, x = 0.2, astext = FALSE, dig = 0)
```

Arguments

pop	vector of population totals across places, like out$results_bysite$pop where out is the output of ejamit()
x	a fraction of 1, the share of all places (or a vector of values)
astext	if TRUE, return text of description of results
dig	rounding digits for text output

Value

A fraction of 1 (or a vector of results) or text

Examples

```
x <- testdata_ejamit_output_100pts_1miles$results_bysite
popshare_p_lives_at_what_pct(x$pop, p = 0.50, astext=TRUE)
popshare_p_lives_at_what_n(  x$pop, p = c(0.50, 0.67, 0.80, 0.95))
popshare_at_top_x_pct(       x$pop, x = c(0.25, 0.50, .90))
popshare_at_top_n(           x$pop, n = c(1, 5, 10))
```

---

popshare_p_lives_at_what_n

popshare_p_lives_at_what_n - how many sites account for P percent of residents?

Description

popshare_p_lives_at_what_n - how many sites account for P percent of residents?

Usage

```
popshare_p_lives_at_what_n(pop, p, astext = FALSE, dig = 0)
```

Arguments

pop	vector of population totals across places, like out$results_bysite$pop where out is the output of ejamit()
p	share of population (0-1, fraction), vector of one or more
astext	if TRUE, return text of description of results
dig	rounding digits for text output

Value

vector of numbers of sites, or text about that

Examples

```
x <- testdata_ejamit_output_100pts_1miles$results_bysite
popshare_p_lives_at_what_pct(x$pop, p = 0.50, astext=TRUE)
popshare_p_lives_at_what_n(  x$pop, p = c(0.50, 0.67, 0.80, 0.95))
popshare_at_top_x_pct(       x$pop, x = c(0.25, 0.50, .90))
popshare_at_top_n(           x$pop, n = c(1, 5, 10))
```

popshare_p_lives_at_what_pct *popshare_p_lives_at_what_pct - what percent of sites account for P percent of residents?*

Description

popshare_p_lives_at_what_pct - what percent of sites account for P percent of residents?

Usage

```
popshare_p_lives_at_what_pct(pop, p, astext = FALSE, dig = 0)
```

Arguments

pop	vector of population totals across places, like out$results_bysite$pop where out is the output of ejamit()
p	share of population (0-1, fraction), vector of one or more
astext	if TRUE, return text of description of results
dig	rounding digits for text output

Value

vector of fractions 0-1 of all sites, or text about that

Examples

```
x <- testdata_ejamit_output_100pts_1miles$results_bysite
popshare_p_lives_at_what_pct(x$pop, p = 0.50, astext=TRUE)
popshare_p_lives_at_what_n(  x$pop, p = c(0.50, 0.67, 0.80, 0.95))
popshare_at_top_x_pct(       x$pop, x = c(0.25, 0.50, .90))
popshare_at_top_n(           x$pop, n = c(1, 5, 10))
```

popup_from_any *popup_from_any - Simple map popup from a data.table or data.frame, one point per row Creates popup vector that leaflet::addCircles or leaflet::addPopups can use. Works similarly to EJAMejscreenapi::popup_from_df, but now extends to data.table*

Description

popup_from_any - Simple map popup from a data.table or data.frame, one point per row Creates popup vector that leaflet::addCircles or leaflet::addPopups can use. Works similarly to EJAMejscreenapi::popup_from_df, but now extends to data.table

Usage

```
popup_from_any(
  x,
  column_names = names(x),
  labels = column_names,
  n = "all",
  testing = FALSE
)
```

Arguments

x,	a data table or data frame
column_names	default is all, or a vector of column names from x to use. If some of column_names requested are not found in names(x), a warning is given and NA values returned for those names not in x. If some of names(x) not requested by column_names, they are left out.
labels	default is column_names - vector used to label the elements in the popup. Must be same length as column_names
n	Show the first n columns of mypoints, in popup. "all" means all of them.
testing	can set to TRUE while testing function

Details

Each popup is made from one row of the data.frame. Each popup has one row of text per column of the data.frame

Value

A vector of strings, one per row or map point, with a line break separating column elements

Examples

```
dat <- data.table(
  RegistryId = c("110071102551", "110015787683"),
  FacilityName = c("USDOI FWS AK MARITIME NWR etc", "ADAK POWER PLANT"),
  LocationAddress = c("65 MI W. OF ADAK NAVAL FACILITY", "100 HILLSIDE BLVD"),
  CityName = c("ADAK", "ADAK"),
  CountyName = c("ALEUTIAN ISLANDS", "ALEUTIANS WEST"),
  StateAbbr = c("AK", "AK"),
  ZipCode = c("99546", "99546"),
  FIPSCode = c("02010", "02016"),
  lat = c(51.671389,51.8703), lon = c(-178.051111, -176.659),
  SupplementalLocation = c(NA_character_,NA_character_))

## add popups only
leaflet::leaflet(dat) |> leaflet::addTiles() |> leaflet::addPopups(popup = popup_from_any(dat))

## add circles with clickable popups
leaflet::leaflet(dat) |> leaflet::addTiles() |> leaflet::addCircles(popup = popup_from_any(dat))

## convert to data frame, works the same way
dat_df <- as.data.frame(dat_df)
leaflet::leaflet(dat) |> leaflet::addTiles() |> leaflet::addCircles(popup = popup_from_any(dat))
```

`proximity.score.in.miles` *proximity.score.in.miles - convert EJScreen proximity scores to miles per site instead of sites per kilometer Shows US percentiles if no arguments used*

Description

proximity.score.in.miles - convert EJScreen proximity scores to miles per site instead of sites per kilometer Shows US percentiles if no arguments used

Usage

```
proximity.score.in.miles(scoresdf = NULL)
```

Arguments

`scoresdf`	data.frame of simple proximity scores like for tsdf, rmp, npl but not traffic.score or npdes one since those are weighted and not just count per km

`proxistat2` *proxistat2 - Calculate a proximity score for every blockgroup - WORK IN PROGRESS Indicator of proximity of each blockgroups to some set of facilities or sites.*

Description

proxistat2 - Calculate a proximity score for every blockgroup - WORK IN PROGRESS Indicator of proximity of each blockgroups to some set of facilities or sites.

Usage

```
proxistat2(pts, countradius = 8.04672, maxradius = 621.3712, quadtree = NULL)
```

Arguments

`pts`	data.table of lat lon
`countradius`	distance within in which nearby sites are counted to create proximity score. In miles, and default is 5km (8.04672 miles) which is the EJScreen zone for proximity scores based on counts.
`maxradius`	max distance in miles to search for nearest single facility, if none found within countradius. EJScreen seems to use 1,000 km as the max to search, since the lowest scores for proximity scores of RMP, TSDF, or NPL are around 0.001, meaning approx. 1/1000 km and km_per_mile = 1.609344 so 1000 km is 1000 / 1.609344 = 621.3712 miles
`quadtree`	must be called localtree, an index of block locations, built during use of EJAM package. see quaddata

Details

Proximity score is sum of (1/d) where each d is distance of a given site in km, summed over all sites within 5km, as in EJScreen.

getblocksnearbyviaQuadTree.R() and maybe doaggregate()?

has a bit of code in it to do some of what this function does.

Value

data.table with proximityscore, bgfips, lat, lon, etc.

Examples

```
# pts <- testpoints_100
# x <- proxistat2(pts = pts[1:1000,], quadtree = localtree)
#
# summary(x$proximityscore)
# # analyze.stuff   pctiles(x$proximityscore)
# plot(x = x$lon, y = x$lat)
# tops = x$proximityscore > 500 & !is.infinite(x$proximityscore) & !is.na(x$proximityscore)
# points(x = x$lon[tops], y = x$lat[tops], col="red")
```

---

quaddata *quaddata (DATA) data.table used to create index of all US block point locations*

Description

quaddata (DATA) data.table used to create index of all US block point locations

Details

8,174,955 rows when non-populated blocks are kept. 5,806,512 rows have Census 2020 population (and blockwt) > 0. This is the largest file used by the package, and is 168 MB as a file, for 2020 Census. - blockid - BLOCK_X, BLOCK_Y, BLOCK_Z (not lat, lon)

```
localtree is the index made from quaddata
(QuadTree class, via SearchTrees pkg), not a data.table
```

See Also

indexblocks() EJAM

radius_inferred *radius_inferred - utility to estimate original radius requested in getblocksnearby() if we only have the outputs of getblocksnearby()*

Description

radius_inferred - utility to estimate original radius requested in getblocksnearby() if we only have the outputs of getblocksnearby()

Usage

```
radius_inferred(
  s2b = NULL,
  decimalsreported = 2,
  decimalsforinferring = 3,
  pctile_of_sites = 0.9,
  nth_furthest_block = 2
)
```

Arguments

s2b	data.table of siteid, distance, etc. that is the output of getblocksnearby()
decimalsreported	parameter to fine tune estimates - generally should not be changed
decimalsforinferring	parameter to fine tune estimates - generally should not be changed
pctile_of_sites	parameter to fine tune estimates - generally should not be changed
nth_furthest_block	parameter to fine tune estimates - generally should not be changed

Details

There are some cases where someone using EJAM functions like getblocksnearby() might in a later separate step use the results of getblocksnearby() to summarize indicator values using a function like doaggregate(), and the actual radius originally requested is not known.

This function tries to approximate what radius must have been requested for analysis, looking at the sites2blocks information about distances to all nearby blocks near each of the analyzed sites. It is not as simple as using the max distance over all sites, because at some sites getblocksnearby() reports one or two distances larger than radius requested, even if avoidorphans is FALSE. That must be because the reported distance is adjusted when it is small relative to the whole block, to better estimate distance to average resident in the block rather than reporting distance to the point that is the block internal point (centroid essentially). As documented in the EJScreen information about creating proximity scores, a facility exactly on top of the block internal point has distance zero to the point but that is not the actual distance to the average resident in the block, hence the adjustment. Some blocks in low density areas are huge so a relatively small circular buffer (small radius) will require adjustments more often. If the block is 3 miles in radius but someone wants a radius of 1 mile in getblocksnearby() or ejamit() analysis overall, a site inside the block might be reported as having a distance of 2.7 miles because the average resident in the block is estimated to be 2.7 miles

away from any site in the block. Almost 2% of US blocks are affected by this issue for a selected radius of 1 mile, but only 1 in 1,000 are for a radius of 3 miles.

This function is based largely on a practical algorithm that is accurate to within 0.01 miles the vast majority of the time for a radius of 1 to 3 miles.

Value

a single number such as 1.5 or 3 that is the estimate of the miles distance that was originally requested in getblocksnearby()

Examples

```
radius_inferred()
 # radius_inferred(getblocksnearby(testpoints_n(100), radius = 3.25))
```

---

regionstats *regionstats (DATA) (obsolete) data.table of 100 percentiles and means for each EPA Region.*

Description

data.table of 100 percentiles and means for each EPA Region (> 1,000 rows) for all the block groups in that zone (e.g., block groups in blockgroupstats) for a set of indicators such as percent low income. Each column is one indicator (or specifies the percentile).

This should be similar to the lookup tables in the gdb on the FTP site of EJScreen.

---

rmost *rmost - utility to rm(list=ls()) but not remove key datasets EJAM uses*

Description

rmost - utility to rm(list=ls()) but not remove key datasets EJAM uses

Usage

```
rmost(
 notremove = c("rmost", "localtree", "blockgroupstats", "usastats", "statestats",
    "bgid2fips", "blockid2fips", "blockpoints", "blockwts", "quaddata", "bgej")
)
```

run_app *run_app - Launch the Shiny Application in RStudio*

Description

launch Shiny web app from RStudio

Usage

```
run_app(
  onStart = NULL,
  options = list(),
  enableBookmarking = "server",
  uiPattern = "/",
  ...
)
```

Arguments

onStart	A function that will be called before the app is actually run. This is only needed for shinyAppObj, since in the shinyAppDir case, a global.R file can be used for this purpose.
options	Named options that should be passed to the runApp call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify width and height parameters which provide a hint to the embedding environment about the ideal height/width for the app.
enableBookmarking	Can be one of "url", "server", or "disable". The default value, NULL, will respect the setting from any previous calls to enableBookmarking(). See enableBookmarking() for more information on bookmarking your app.
uiPattern	A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
...	arguments to pass to golem_opts. Maybe could be something like sitepoints="latlondata.xlsx" or sitepoints=testpoints_100 See ?golem::get_golem_options for more details.

Details

app_run_EJAM() is like run_app() from the EJAM package

`setdiff2` *setdiff2 aka unshared - UTILITY - see what is only in x or y but not both utility just like setdiff except for y,x and also x,y Just shows which elements are in one and only one of the sets x and y*

Description

setdiff2 aka unshared - UTILITY - see what is only in x or y but not both utility just like setdiff except for y,x and also x,y Just shows which elements are in one and only one of the sets x and y

Usage

```
setdiff2(x, y)
```

`setdiff_yx` *setdiff_yx - UTILITY - see what is in y not x utility just like setdiff except for y,x instead of x,y*

Description

setdiff_yx - UTILITY - see what is in y not x utility just like setdiff except for y,x instead of x,y

Usage

```
setdiff_yx(x, y)
```

`shapefile_clean` *shapefile_clean - drop invalid rows and warn if all invalid*

Description

shapefile_clean - drop invalid rows and warn if all invalid

Usage

```
shapefile_clean(shp, crs = 4269)
```

Arguments

`shp`	a shapefile object using sf::read_sf()
`crs`	used in shp <- sf::st_transform(shp, crs = crs), default is crs = 4269 or Geodetic CRS NAD83

Value

a shapefile object using sf::read_sf()

See Also

`shapefile_from_folder()`

shapefile_filepaths_from_folder

shapefile_filepaths_from_folder - get list of valid filenames comprising shapefile including paths

Description

shapefile_filepaths_from_folder - get list of valid filenames comprising shapefile including paths

Usage

```
shapefile_filepaths_from_folder(folder = NULL)
```

Arguments

folder	path of folder that contains the files (.shp, .shx, .dbf, and .prj)

Value

string vector of filenames including full paths

See Also

shapefile_from_folder()

shapefile_filepaths_valid

shapefile_filepaths_valid - confirm files have all the extensions .shp, .shx, .dbf, and .prj

Description

shapefile_filepaths_valid - confirm files have all the extensions .shp, .shx, .dbf, and .prj

Usage

```
shapefile_filepaths_valid(filepaths)
```

Arguments

filepaths	vector of full paths with filenames (types .shp, .shx, .dbf, and .prj) as strings

Value

logical, indicating if all 4 extensions are found among the filepaths

See Also

shapefile_from_folder()

shapefile_from_filepaths *shapefile_from_filepaths - Read shapefile from disk based on the filenames given*

Description

shapefile_from_filepaths - Read shapefile from disk based on the filenames given

Usage

```
shapefile_from_filepaths(filepaths = NULL, cleanit = TRUE, crs = 4269)
```

Arguments

filepaths	vector of full paths with filenames (types .shp, .shx, .dbf, and .prj) as strings
cleanit	set to FALSE if you want to skip validation and dropping invalid rows
crs	if cleanit = TRUE, crs is passed to shapefile_clean() default is crs = 4269 or Geodetic CRS NAD83 Also can check this via x <- sf::st_crs(sf::read_sf()); x$input

Value

a shapefile object using sf::read_sf()

See Also

shapefile_from_folder()

shapefile_from_folder *shapefile_from_folder - read shapefile from a folder*

Description

shapefile_from_folder - read shapefile from a folder

Usage

```
shapefile_from_folder(folder = NULL, cleanit = TRUE, crs = 4269)
```

Arguments

folder	path of folder that contains the files (.shp, .shx, .dbf, and .prj)
cleanit	set to FALSE if you want to skip validation and dropping invalid rows
crs	passed to shapefile_from_filepaths() default is crs = 4269 or Geodetic CRS NAD83

Value

a shapefile object using sf::read_sf()

Examples

```
## Not run:
  testfolder <- system.file("testdata/shapes/Portland_neighborhoods", package = "EJAM")
  testshape <- shapefile_from_folder(testfolder)

  testpaths <- shapefile_filepaths_from_folder(testfolder)
  testshape <- shapefile_from_filepaths(testpaths)

  ## if interactive(), R user can point to right folder or select the right set of files:
  # testshape <- shapefile_from_filepaths()
  # testshape <- shapefile_from_folder()

  x <- get_blockpoints_in_shape(testshape)
  leaflet(x$polys) %>% addTiles() %>% addPolygons(color = "blue")
  DT::datatable(out$results_bysite)


## End(Not run)
```

---

shapefile_from_sitepoints

shapefile_from_sitepoints - convert table of lat,lon points/sites into sf:: shapefile Creates a simple feature (sf) dataframe from points

---

Description

shapefile_from_sitepoints - convert table of lat,lon points/sites into sf:: shapefile Creates a simple feature (sf) dataframe from points

Usage

```
shapefile_from_sitepoints(sitepoints, crs = 4269)
```

Arguments

sitepoints	a data.table or data.frame with columns called lat,lon
crs	used in st_as_sf() default is crs = 4269 or Geodetic CRS NAD83

Value

A shapefile via sf::st_as_sf()

See Also

get_blockpoints_in_shape() shapefile_from_sitepoints() shape_buffered_from_shapefile_points()

shapes_blockgroups_from_bgfips

use API to get boundaries of blockgroups

Description

use API to get boundaries of blockgroups

Usage

```
shapes_blockgroups_from_bgfips(
  bgfips = "010890029222",
  outFields = "",
  myservice =
   c("https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/USA_Boundaries_2022/Feat
   "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/USA_Block_Groups/FeatureSe
   "https://services.arcgis.com/cJ9YHowT8TU7DUyn/ArcGIS/rest/services/EJScreen_2_21_US_Percentil
)
```

Arguments

bgfips	one or more block group FIPS codes as 12-character strings in a vector
outFields	can be "*" for all, or can be just some variables like SQMI, POPULATION_2020, etc., or none
myservice	URL of feature service to get shapes from. "https://services.arcgis.com/cJ9YHowT8TU7DUyn/ArcGIS/rest/services/ EJScreen_2_21_US_Perc for example provides EJScreen indicator values, NPL_CNT, TSDF_CNT, EXCEED_COUNT_90, etc.

Details

This is useful mostly for small numbers of blockgroups. The EJScreen map services provide other ways to map blockgroups and see EJScreen data.

Value

spatial object via sf::read_sf()

shapes_counties_from_countyfips

use API to get boundaries of US Counties to map them

Description

use API to get boundaries of US Counties to map them

Usage

```
shapes_counties_from_countyfips(
  countyfips = "10001",
  outFields = "",
  myservice =
   c("https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/USA_Boundaries_2022/Feat
   "https://services.arcgis.com/P3ePLMYs2RVChkJx/ArcGIS/rest/services/USA_Counties_and_States_wi
   "https://services.arcgis.com/cJ9YHowT8TU7DUyn/ArcGIS/rest/services/EJScreen_2_22_US_Percentil
)
```

Arguments

countyfips	FIPS codes as 5-character strings (or numbers) in a vector
outFields	can be "*" for all, or can be just some variables like SQMI, POPULATION_2020, etc., or none
myservice	URL of feature service to get shapes from. Only default was tested

Value

spatial object via sf::read_sf()

---

shape_buffered_from_shapefile

shape_buffered_from_shapefile - add buffer around shape

Description

shape_buffered_from_shapefile - add buffer around shape

Usage

```
shape_buffered_from_shapefile(shapefile, radius.miles, crs = 4269, ...)
```

Arguments

shapefile	spatial object like areas at high risk or areas with facilities to be analyzed
radius.miles	width of buffer to add to shapefile (in case dist is a units object, it should be convertible to arc_degree if x has geographic coordinates, and to st_crs(x)$units otherwise)
crs	used in st_transform() default is crs = 4269 or Geodetic CRS NAD83
...	passed to st_buffer()

Details

Just a wrapper for `sf::st_buffer()`

See Also

`get_blockpoints_in_shape()` `shapefile_from_sitepoints()` shape_buffered_from_shapefile_points

shape_buffered_from_shapefile_points

shape_buffered_from_shapefile_points - add buffer around shape (points, here)

Description

shape_buffered_from_shapefile_points - add buffer around shape (points, here)

Usage

```
shape_buffered_from_shapefile_points(
  shapefile_points,
  radius.miles,
  crs = 4269,
  ...
)
```

Arguments

shapefile_points	spatial object like areas at high risk or areas with facilities to be analyzed
radius.miles	width of buffer to add to shapefile_points (in case dist is a units object, it should be convertible to arc_degree if x has geographic coordinates, and to st_crs(x)$units otherwise)
crs	used in st_transform() default is crs = 4269 or Geodetic CRS NAD83
...	passed to st_buffer()

Details

Just a wrapper for `sf::st_buffer()`

See Also

`get_blockpoints_in_shape()` `shapefile_from_sitepoints()` shape_buffered_from_shapefile_points

SIC

SIC (DATA) named list of all SIC code numbers and category name for each

Description

SIC (DATA) named list of all SIC code numbers and category name for each

Details

see https://siccode.com

See Also

SIC sictable `sic_categories()`

`sictable` — *sictable (DATA) data.table of SIC code(s) and industry names for each EPA-regulated site in Facility Registry Service Also has the 2,3, and 4-digit SIC that this code falls under, where relevant for given length*

Description

This is similar to the data file EJAM::SIC but in a more useful format and newer functions work with it.

Details

see https://siccode.com

See Also

SIC sictable `sic_categories()`

`sic_categories` — *sic_categories - See the names of SIC industrial categories and their codes Easy way to view, in RStudio console, the SIC categories. SIC all are 4-digit codes, like 7218 - Industrial launderers*

Description

sic_categories - See the names of SIC industrial categories and their codes Easy way to view, in RStudio console, the SIC categories. SIC all are 4-digit codes, like 7218 - Industrial launderers

Usage

```
sic_categories()
```

See Also

SIC naics_categories

sic_from_any	*sic_from_any - General way to search for industry names and NAICS codes Find industry names and codes by searching for queried code(s) or text*

Description

sic_from_any - General way to search for industry names and NAICS codes Find industry names and codes by searching for queried code(s) or text

Usage

```
sic_from_any(
  query,
  children = FALSE,
  ignore.case = TRUE,
  fixed = FALSE,
  website_scrape = FALSE,
  website_url = FALSE
)
```

Arguments

query	query string(s) and/or number(s), vector of NAICS codes or industry names or any regular expression or partial words
children	logical, if TRUE, also return all the subcategories - where NAICS starts with the same digits
ignore.case	see grepl()
fixed	should it be an exact match? see grepl()
website_scrape	whether to scrape info from the NAICS website to return a table of codes and names that match (web query uses synonyms so gets more hits)
website_url	whether to return the URL of the webpage with info on the NAICS (web query uses synonyms so gets more hits)

Value

a subset of the sictable data.table (not just the codes column)

See Also

sic_subcodes_from_code() sic_from_code() sic_from_name() sic_from_any()

`sic_from_code` *sic_from_code - search for industry names by SIC code(s), 4 digits each*

Description

sic_from_code - search for industry names by SIC code(s), 4 digits each

Usage

```
sic_from_code(mycodes, children = FALSE)
```

Arguments

`mycodes`	vector of character SIC codes. see https://siccode.com
`children`	logical, if TRUE, also return all the subcategories - where SIC starts with the same digits

Value

a subset of the sictable data.table (not just the codes column)

See Also

`sic_subcodes_from_code()` `sic_from_code()` `sic_from_name()`

`sic_from_name` *sic_from_name - search for industry names and SIC codes by query string query by parts of words, etc. in the industry name.*

Description

sic_from_name - search for industry names and SIC codes by query string query by parts of words, etc. in the industry name.

Usage

```
sic_from_name(mynames, children = FALSE, ignore.case = TRUE, fixed = FALSE)
```

Arguments

`mynames`	query string, vector of SIC industry names or any regular expression or partial words. See https://siccode.com
`children`	logical, if TRUE, also return all the subcategories - where SIC starts with the same digits
`ignore.case`	see `grepl()`
`fixed`	should it be an exact match? see `grepl()`

Value

a subset of the sictable data.table (not just the codes column)

See Also

sic_subcodes_from_code() sic_from_code() sic_from_name() sic_from_any()

Examples

```
data.table::fintersect(sic_from_any( "glass"), sic_from_any("paint"))
```

`sic_subcodes_from_code`

sic_subcodes_from_code - find subcategories of the given overall SIC industry code(s) Given 3-digit SIC code, for example, get all SIC that start with those digits.

Description

sic_subcodes_from_code - find subcategories of the given overall SIC industry code(s) Given 3-digit SIC code, for example, get all SIC that start with those digits.

Usage

```
sic_subcodes_from_code(mycodes)
```

Arguments

mycodes	SIC codes vector, of 2 to 4 digits each. See https://siccode.com

Details

similar idea was naics2children() but this is more robust See sic_from_any() which uses this

Value

a subset of the sictable data.table (not just the codes column)

See Also

sic_subcodes_from_code() sic_from_code() sic_from_name() sic_from_any()

Examples

```
# codes starting with '07'
sic_subcodes_from_code('07')
# codes starting with '078'
sic_subcodes_from_code('078')
```

siteid_from_naics *latlon_from_naics - Find EPA-regulated facilities in FRS by NAICS code (industrial category) Get lat lon, Registry ID, given NAICS industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact NAICS code (not subcategories)*

Description

latlon_from_naics - Find EPA-regulated facilities in FRS by NAICS code (industrial category) Get lat lon, Registry ID, given NAICS industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact NAICS code (not subcategories)

Usage

```
siteid_from_naics(naics, id_only = FALSE)
```

Arguments

naics	a vector of naics codes, or a data.table with column named code, as with output of naics_from_any()

Details

NOTE: many FRS sites lack NAICS code!

Also, this function does not find the sites identified by FRS data as being in a child NAICS (subcategory of your exact query)!

Relies on frs_by_naics (a data.table)

See info about NAICS industry codes at https://www.naics.com/search

Value

A data.table (not just data.frame) with columns called lat, lon, REGISTRY_ID, NAICS (but see the id_only parameter)

Examples

```
siteid_from_naics(321114)
latlon_from_naics(321114)
latlon_from_naics(EJAM::naics_from_any("cheese")[,code] )
head(latlon_from_naics(c(3366, 33661, 336611), id_only=TRUE))
# mapfast(frs_from_naics(336611)) # simple map
```

siteid_from_sic *latlon_from_sic - Find EPA-regulated facilities in FRS by SIC code (industrial category)*

Description

Get lat lon, Registry ID, given SIC industry code(s) Find all EPA Facility Registry Service (FRS) sites with this exact SIC code (not subcategories)

Usage

```
siteid_from_sic(sic, id_only = FALSE)
```

Arguments

sic	a vector of SIC codes, or a data.table with column named code, as with output of sic_from_any()

Details

NOTE: many FRS sites lack SIC code!

Also, this function does not find the sites identified by FRS data as being in a child SIC (subcategory of your exact query)!

Relies on frs_by_sic (a data.table)

See info about SIC industry codes at https://www.naics.com/search

Value

A data.table (not just data.frame) with columns called lat, lon, REGISTRY_ID, SIC (but see the id_only parameter)

Examples

```
siteid_from_sic('7300')
latlon_from_sic('7300')
latlon_from_sic(sic_from_any("cheese")[,code] )
head(latlon_from_sic(c('6150', '6300', '5995'), id_only=TRUE))
# mapfast(frs_from_sic('6150')) # simple map
```

sites2blocks_example1000pts_1miles *test output of getblocksnearby(), and is an input to doaggregate()*

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_1000, radius = 1) This is the same as testoutput_getblocksnearby_1000pts_1miles

See Also

getblocksnearby() doaggregate() testpoints_1000

---

sites2blocks_example100pts_1miles

test output of getblocksnearby(), and is an input to doaggregate()

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_100, radius = 1) This is the same as testoutput_getblocksnearby_100pts_1miles

See Also

getblocksnearby() doaggregate() testpoints_100

---

sites2blocks_example10pts_1miles

test output of getblocksnearby(), and is an input to doaggregate()

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_10, radius = 1) This is the same as testoutput_getblocksnearby_10pts_1m

See Also

getblocksnearby() doaggregate() testpoints_10

speedtable_expand *speedtable_expand Utility used by speedtest() and speedtable_summarize()*

Description

speedtable_expand Utility used by speedtest() and speedtable_summarize()

Usage

```
speedtable_expand(speedtable)
```

Arguments

speedtable	must have columns called points, miles, and perhr

speedtable_summarize *speedtable_summarize utility used by speedtest()*

Description

speedtable_summarize utility used by speedtest()

Usage

```
speedtable_summarize(speedtable)
```

Arguments

speedtable	from speedtest(), with columns named points and perhr

See Also

speedtest()

speedtest *speedtest Run EJAM analysis for several radii and numbers of sitepoints, recording how long each step takes*

Description

speedtest Run EJAM analysis for several radii and numbers of sitepoints, recording how long each step takes

Usage

```
speedtest(
  n = 10,
  sitepoints = NULL,
  weighting = "frs",
  radii = c(1, 3.106856, 5, 10, 31.06856)[1:3],
  avoidorphans = FALSE,
  test_ejamit = FALSE,
  test_getblocksnearby = TRUE,
  test_doaggregate = TRUE,
  test_batch.summarize = FALSE,
  logging = FALSE,
  logfolder = getwd(),
  logfilename = "log_n_datetime.txt",
  honk_when_ready = TRUE,
  saveoutput = FALSE,
  plot = TRUE,
  getblocks_diagnostics_shown = FALSE,
  ...
)
```

Arguments

n	optional, vector of 1 or more counts of how many random points to test, or set to 0 to interactively pick file of points in RStudio (n is ignored if sitepoints provided)
sitepoints	optional, (use if you do not want random points) data.frame of points or path/file with points, where columns are lat and lon in decimal degrees
weighting	optional, if using random points, how to weight them, such as facilities, people, or blockgroups. see testpoints_n()
radii	optional, one or more radius values in miles to use in creating circular buffers when findings residents nearby each of sitepoints. The default list includes one that is 5km (approx 3.1 miles)
avoidorphans	see getblocksnearby() or ejamit() regarding this param
test_ejamit	whether to test only ejamit() instead of its subcomponents like getblocksnearby(), doaggregate(), etc
test_getblocksnearby	whether to include this function in timing - not used because always done

test_doaggregate
: whether to include this function in timing

test_batch.summarize
: whether to include this function in timing

logging
: logical optional, whether to save log file with timings of steps. NOTE this slows it down though.

logfolder
: optional, name of folder for log file

logfilename
: optional, name of log file to go in folder

honk_when_ready
: optional, self-explanatory

saveoutput
: but this slows it down if set to TRUE to save each run as .rda file

plot
: whether to create plot of results

getblocks_diagnostics_shown
: set TRUE to see more details on block counts etc.

...
: passed to plotting function

Details

This is essentially a test script that times each step of EJAM for a large dataset

- pick a sample size (n) (or enter sitepoints, or set n=0 to interactively pick file of points in RStudio)
- pick n random points
- pick a few different radii for circular buffering
- analyze indicators in circular buffers and overall (find blocks nearby and then calc indicators)
- get stats that summarize those indicators
- compare times between steps and radii and other approaches or tools

Value

EJAM results similar to as from the web app or `ejamit()` and also creates a plot

See Also

`speedtest_plot()`

Examples

```
## Not run:
  speedseen_few <- speedtest(c(50,500), radii=c(1, 3.106856), logging=FALSE, honk=FALSE)

  speedseen_nearer_to1k <- speedtest(n = c(1e2,1e3,1e4 ), radii=c(1, 3.106856,5 ),
    logging=TRUE, honk=FALSE)
  save( speedseen_nearer_to1k, file = "~/../Downloads/speedseen_nearer_to1k.rda")
  rstudioapi::savePlotAsImage(        "~/../Downloads/speedseen_nearer_to1k.png")

  speedseen_all <- speedtest(
    n = c(1e2,1e3,1e4),
    radii=c(1, 3.106856, 5, 10, 31.06856),
    logging=TRUE, honk=TRUE
  )

## End(Not run)
```

`speedtest_plot` *speedtest_plot utility to plot output of speedtest(), rate of points analyzed per hour*

Description

speedtest_plot utility to plot output of speedtest(), rate of points analyzed per hour

Usage

```
speedtest_plot(x, ltype = "b", plotfile = NULL, secondsperthousand = FALSE)
```

Arguments

`x`	table from speedtest()
`ltype`	optional type of line for plot
`plotfile`	optional path and filename of .png image file to save

Value

side effect is a plot. returns x but with seconds column added to it

See Also

`speedtest()`

`stateinfo` *stateinfo (DATA) data.frame of state abbreviations and state names (50+DC+PR; not AS, GU, MP, VI, UM)*

Description

52 rows and a few variables: ST is the 2-letter abbreviation, statename is the State name, etc.

Details

Created for EJAM by datacreate_stateinfo.R script

Also see https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/release/2020-island-areas-data-products.html

column names: "ST" "statename" "ftpname" "FIPS.ST" "REGION"

Some datasets lack PR. (72)

Many datasets lack these: AS, GU, MP, VI (codes "60" "66" "69" "78")

Almost all datasets lack UM. (74)

```
72 PR                 Puerto Rico

66 GU                        Guam

69 MP    Northern Mariana Islands

78 VI         U.S. Virgin Islands

74 UM U.S. Minor Outlying Islands
```

See Also

stateinfo2 for more columns

`stateinfo2` *stateinfo2 (DATA) data.frame of state abbreviations and state names (50+DC+PR; not AS, GU, MP, VI, UM)*

Description

52 rows and several variables: ST is the 2-letter abbreviation, statename is the State name, etc.

Details

Created for EJAM by datacreate_stateinfo2.R script

Also see https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/release/2020-island-areas-data-products.html

column names:

c("statename", "FIPS.ST", "ST", "ftpname", "REGION",

"is.usa.plus.pr", "is.usa", "is.state", "is.contiguous.us", "is.island.areas",

"area.sqmi", "area.sqkm",

"landarea.sqmi", "landarea.sqkm",

"waterarea.sqmi", "waterarea.sqkm",

"inland.sqmi", "inland.sqkm",

"coastal.sqmi", coastal.sqkm",

"greatlakes.sqmi", "greatlakes.sqkm",

"territorial.sqmi", "territorial.sqkm",

"lat", "lon")

Some datasets lack PR. (72)

Many datasets lack these: AS, GU, MP, VI (codes "60" "66" "69" "78")

Almost all datasets lack UM. (74)

```
72 PR                 Puerto Rico

66 GU                        Guam
```

```
69 MP    Northern Mariana Islands

78 VI         U.S. Virgin Islands

74 UM U.S. Minor Outlying Islands
```

See Also

stateinfo for fewer columns

`statestats` *statestats (DATA) data.frame of 100 percentiles and means for each US State and PR and DC.*

Description

data.frame of 100 percentiles and means for each US State and PR and DC (approx 5,300 rows) for all the block groups in that zone (e.g., block groups in blockgroupstats) for a set of indicators such as percent low income. Each column is one indicator (or specifies the percentile).

This should be similar to the lookup tables in the gdb on the FTP site of EJScreen, except it also has data for the demographic race/ethnicity subgroups. For details on how the table was made, see /EJAM/data-raw/usastats_subgroups.R

`statestats_means` *statestats_means - convenient way to see STATE MEANS of ENVIRONMENTAL and DEMOGRAPHIC indicators*

Description

statestats_means - convenient way to see STATE MEANS of ENVIRONMENTAL and DEMOGRAPHIC indicators

Usage

```
statestats_means(
  ST = unique(EJAM::statestats$REGION),
  varnames = c(EJAM::names_e, EJAM::names_d, EJAM::names_d_subgroups_nh),
  PCTILES = "mean",
  dig = 2
)
```

Arguments

ST	vector of state abbreviations, or USA
varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	digits to round to

statestats_query *statestats_query - convenient way to see mean, pctiles of Env or Demog indicators from lookup table*

Description

statestats_query - convenient way to see mean, pctiles of Env or Demog indicators from lookup table

Usage

```
statestats_query(
  ST = sort(unique(EJAM::statestats$REGION)),
  varnames = c(EJAM::names_e, EJAM::names_d),
  PCTILES = NULL,
  dig = 2
)
```

Arguments

ST	vector of state abbreviations, or USA
varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	digits to round to

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us               # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector

usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))
```

```
## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')

## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)), .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

`statestats_queryd` *statestats_queryd - convenient way to see mean, pctiles of DEMOG indicators from lookup table*

Description

statestats_queryd - convenient way to see mean, pctiles of DEMOG indicators from lookup table

Usage

```
statestats_queryd(
  ST = sort(unique(EJAM::statestats$REGION)),
  varnames = c(EJAM::names_d, EJAM::names_d_subgroups_nh),
  PCTILES = NULL,
```

```
  dig = 2
)
```

Arguments

ST	vector of state abbreviations, or USA
varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	digits to round to

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us                # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector

usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))

## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')
```

```
## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)), .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

`statestats_querye` *statestats_querye - convenient way to see mean, pctiles of ENVIRONMENTAL indicators from lookup table*

Description

statestats_querye - convenient way to see mean, pctiles of ENVIRONMENTAL indicators from lookup table

Usage

```
statestats_querye(
  ST = sort(unique(EJAM::statestats$REGION)),
  varnames = EJAM::names_e,
  PCTILES = NULL,
  dig = 2
)
```

Arguments

`dig`	how many digits to round to

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us               # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector
```

```
usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))

## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')

## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),  .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

states_infer — *states_infer - Get table of info on States (from latlon or FIPS) Get cleaned table of US State etc. by siteid, from lat/lon, or from FIPS*

Description

states_infer - Get table of info on States (from latlon or FIPS) Get cleaned table of US State etc. by siteid, from lat/lon, or from FIPS

Usage

```
states_infer(x)
```

Arguments

x	data.frame or data.table with either ST column or lat and lon columns, or FIPS, and optionally a column with siteid or column called n

Value

data.frame with unique siteid, ST, etc.

See Also

state_from_latlon() state_from_fips()

states_shapefile — *states_shapefile (DATA) US States boundaries 2020 shapefile from TIGER*

Description

This is used to figure out which state contains each point (facility/site).

Details

This is used by state_from_latlon() to find which state is associated with each point that the user wants to analyze. That is needed to report indicators in the form of State-specific percentiles (e.g., a score that is at the 80th percentile within Texas). It is created by the package via a script at EJAM/data-raw/datacreate_states_shapefile.R which downloads the data from Census Bureau.

See Also

state_from_latlon() get_blockpoints_in_shape()

`state_from_blockid` *state_from_blockid given vector of blockids, get state abbreviation of each*

Description

state_from_blockid given vector of blockids, get state abbreviation of each

Usage

```
state_from_blockid(blockid)
```

Arguments

`blockid`	vector of blockid values as from EJAM in a table called blockpoints

Value

vector of ST info like AK, CA, DE, etc.

Examples

```
state_from_blockid(c(8174952, blockpoints$blockid[5:6]))
```

`state_from_blocktable` *state_from_blocktable - was used only in some special cases of using testpoints_n() given data.table with blockid column, get state abbreviation of each - not used?*

Description

state_from_blocktable - was used only in some special cases of using testpoints_n() given data.table with blockid column, get state abbreviation of each - not used?

Usage

```
state_from_blocktable(dt_with_blockid)
```

Arguments

`dt_with_blockid`

Value

vector of ST info like AK, CA, DE, etc.

Examples

```
state_from_blocktable(blockpoints[45:49,])
```

`state_from_fips` *state_from_fips - Get FIPS of ALL BLOCKGROUPS in the States or Counties Get the State abbreviations of ALL blockgroups within the input FIPS*

Description

state_from_fips - Get FIPS of ALL BLOCKGROUPS in the States or Counties Get the State abbreviations of ALL blockgroups within the input FIPS

Usage

```
state_from_fips(fips, uniqueonly = FALSE)
```

Arguments

`fips`	Census FIPS codes vector, numeric or char, 2-digit, 5-digit, etc. OK
`uniqueonly`	If set to TRUE, returns only unique results. This parameter is here mostly to remind user that default is not uniques only.

Details

Returns a vector of 2-letter State abbreviations that is one per blockgroup that matches the input FIPS, not necessarily a vector as long as the input vector of FIPS codes!, and not just a short list of unique states!

Value

vector of 2-character state abbreviations like CA,CA,CA,MD,MD,TX

`state_from_latlon` *state_from_latlon - find what state is where each point is located Takes 3 seconds to find state for 1k points, so a faster alternative would be useful*

Description

state_from_latlon - find what state is where each point is located Takes 3 seconds to find state for 1k points, so a faster alternative would be useful

Usage

```
state_from_latlon(lat, lon, states_shapefile = EJAM::states_shapefile)
```

Arguments

`lat`	latitudes vector
`lon`	longitudes vector
`shapefile`	shapefile of US States, in package already

Value

Returns data.frame: ST, statename, FIPS.ST, REGION, n as many rows as elements in lat or lon

See Also

states_shapefile get_blockpoints_in_shape() states_infer()

Examples

```
myprogram <- "CAMDBS" # 739 sites
pts <- frs_from_program(myprogram)[ , .(lat, lon, REGISTRY_ID,  PRIMARY_NAME)]
# add a column with State abbreviation
pts[, ST := state_from_latlon(lat=lat, lon = lon)$ST]
#map these points
mapfast(pts[ST == 'TX',], radius = 1) # 1 miles radius circles
```

---

structure.of.output.list

structure.of.output.list - See info about list of results Utility to print summary info about the output of ejamit or doaggregate

---

Description

structure.of.output.list - See info about list of results Utility to print summary info about the output of ejamit or doaggregate

Usage

```
structure.of.output.list(x, maxshown = 10)
```

Arguments

x	the output of ejamit() or of doaggregate(), a list of objects holding results of analysis
maxshown	shows only first 10 elements of list by default

Value

data.frame summarizing names of list, whether each element is a data.table, data.frame, or vector, and rows/cols/length info

Examples

```
structure.of.output.list(testpoints_10)
structure.of.output.list(testoutput_getblocksnearby_10pts_1miles)
structure.of.output.list(testoutput_doaggregate_10pts_1miles)
structure.of.output.list(testoutput_ejamit_10pts_1miles)
structure.of.output.list(testoutput_ejscreenapi_plus_5)
structure.of.output.list(testoutput_ejscreenit_5)
```

ST_by_site_from_sites2blocks

ST_by_site_from_sites2blocks - Get State that each site is in, from a table of siteid, blockid, distance

Description

Find the 2-character State abbreviation for each site. This is for when you need to know the state each site is in, to be able to report state percentiles, but you do not have the original list of siteid lat/lon or State info. This can infer the State each site is located in, based on the state of the nearest block (and its parent blockgroup).

Usage

```
ST_by_site_from_sites2blocks(sites2blocks)
```

Arguments

sites2blocks	data.table or data.frame, like testoutput_getblocksnearby_10pts_1miles, from getblocksnearby() that has columns siteid and blockid and distance

Value

data.table with columns siteid, ST

Examples

```
## Not run:
 fname = './inst/testdata/testpoints_207_sites_with_signif_violations_NAICS_326_ECHO.csv'
  x = ST_by_site_from_sites2blocks(
    getblocksnearby( latlon_from_anything(fname), quadtree = localtree))
  y = read_csv_or_xl(fname)
  x$ST == y$FacState

## End(Not run)
  ST_by_site_from_sites2blocks(testoutput_getblocksnearby_10pts_1miles)
```

table4gt_from_scorevectors

table4gt_from_scorevectors - DRAFT EXPERIMENTAL - attempt to make table more flexible / any indicators Based on just indicator names and a value for each, it tries to fill in the rest of a summary table's data. and formats this as a data.frame ready for the next step

Description

table4gt_from_scorevectors - DRAFT EXPERIMENTAL - attempt to make table more flexible / any indicators Based on just indicator names and a value for each, it tries to fill in the rest of a summary table's data. and formats this as a data.frame ready for the next step

Usage

```
table4gt_from_scorevectors(
  varnames_r = names_e,
  varnames_shown = fixcolnames(varnames_r, "r", "long"),
  value = as.vector(usastats_means(varnames_r)),
  state_avg = NULL,
  state_pctile = NULL,
  usa_avg = NULL,
  usa_pctile = NULL,
  state_ratio = NULL,
  usa_ratio = NULL,
  ST = "NY"
)
```

Arguments

varnames_r	vector of variable names like names_d
varnames_shown	vector like names_d_friendly
value	indicator values for a place or overall
state_avg	indicator values average in State
state_pctile	indicator values as State percentiles
usa_avg	indicator values US average
usa_pctile	indicator values as US percentiles
state_ratio	indicator values as ratio to State average
usa_ratio	indicator values as ratio to US average

Value

data.frame ready for table_gt_format_step2 ???

See Also

table_gt_from_ejamit() table_gt_from_ejamit_overall() table_gt_from_ejamit_1site()
table_validated_ejamit_row() table_gt_format_step1() table_gt_format_step2()

---

tablefixed *tablefixed - Table of counts of integer values zero through maxbin*

Description

Like tabulate or table, sort of, but includes zero unlike tabulate, and lets you ensure results include every integer 0 through maxbin, so you can, for example, easily combine tables of counts where some did not include all integers.

Usage

```
tablefixed(x, maxbin = NULL)
```

Arguments

x	vector of integers, like counts, that can include 0
maxbin	highest integer among x, or number of bins

Details

There is likely a more efficient way to do this in some existing package, but this is useful and fast enough.

When using a dataset like EJScreen with 13 indicators of interest, and counting how many of the 13 are above various cutpoints, there may be zero rows that have exactly 8 above some cutoff, for example.

This function makes it easier to combine those tables into a summary where 0-13 are in each table while table() would only return integers that came up in a given case (for one cutoff).

Value

summary table

See Also

colcounter_summary()

---

table_gt_format_step1 *table_gt_format_step1 - validate and reshape 1 row of ejamit results to prep for formatting as gt table/report reshapes a few columns of a 1 row data.table into a tall multirow data.frame.*

Description

table_gt_format_step1 - validate and reshape 1 row of ejamit results to prep for formatting as gt table/report reshapes a few columns of a 1 row data.table into a tall multirow data.frame.

Usage

```
table_gt_format_step1(ejamit_results_1row = NULL, type = "demog")
```

Arguments

ejamit_results_1row	data.table (or data.frame) like testoutput_ejamit_100pts_1miles$results_overall from something like ejamit(testpoints_100, radius = 1)$results_overall
type	demog or envt to specify which type of table

See Also

table_gt_from_ejamit() table_gt_from_ejamit_overall() table_gt_from_ejamit_1site() table_validated_ejamit_row() table_gt_format_step1() table_gt_format_step2()

table_gt_format_step2 *table_gt_format_step2 - Format a table of demog or envt scores, percentiles, etc. to look similar to EJScreen report tables*

Description

table_gt_format_step2 - Format a table of demog or envt scores, percentiles, etc. to look similar to EJScreen report tables

Usage

```
table_gt_format_step2(
  df,
  type = c("demog", "envt")[1],
  my_cell_color = "#dce6f0",
  my_border_color = "#aaaaaa",
  digits_default = 2
)
```

Arguments

df	A data frame from table_gt_format_step1 which is just a specific format of key EJAM results. It has these columns (but it still works if the first two are omitted and user-provided indicators are used - it just names them indicator 1, indicator 2, etc.): varnames_r, varnames_shown, value, state_avg, state_pctile, usa_avg, usa_pctile and one row per indicator, where varnames_shown are longer indicator names for use in report. The sort order in this df is ignored! Instead, the variables are shown in the same order as shown in EJScreen reports, as recorded in map_headernames and checked here via varinfo(varnames_r, "reportsort"), etc. Uses gt R package for formatting.
type	string - must be demog or envt
my_cell_color	color for table cell fill backgrounds, can be given as string ('blue') or hex code ('#0070c0')
my_border_color	color for table borders and boundaries, can be given as string ('blue') or hex code ('#0070c0')
digits_default	number of digits to round to if not specified for a given indicator (rounding info is drawn from map_headernames$decimals)

Value

a gt-style table with formatting to closely match EJScreen standard report formatting

See Also

table_gt_from_ejamit()

table_gt_from_ejamit *table_gt_from_ejamit - Create a gt-format table of results from EJAM Uses the list of results of ejamit()*

Description

table_gt_from_ejamit - Create a gt-format table of results from EJAM Uses the list of results of ejamit()

Usage

```
table_gt_from_ejamit(ejamitoutput = NULL, type = c("demog", "envt")[1])
```

Arguments

ejamitoutput	list of EJAM results formatted as in testoutput_ejamit_100pts_1miles, as would be the output of ejamit()
type	Must be "demog" or "envt" – Creates one of these at a time

Details

See the R package called gt. Also see code that creates html tables from html template and code that creates formatted spreadsheets like table_xls_format()

Value

Provides table in gt format from the R package called gt

Examples

```
table_gt_from_ejamit(testoutput_ejamit_100pts_1miles)
```

table_gt_from_ejamit_1site *table_gt_from_ejamit_1site - Create a formatted table of results for 1 site from EJAM Uses 1 row from the results_bysite part of ejamit() output*

Description

table_gt_from_ejamit_1site - Create a formatted table of results for 1 site from EJAM Uses 1 row from the results_bysite part of ejamit() output

Usage

```
table_gt_from_ejamit_1site(...)
```

Arguments

...	passed to table_gt_from_ejamit_overall()

Examples

```
table_gt_from_ejamit_1site(testoutput_ejamit_100pts_1miles$results_bysite[ 1, ])
```

`table_gt_from_ejamit_overall` *table_gt_from_ejamit_overall - Create a formatted table of results from EJAM overall summary stats Uses the results_overall element of ejamit() output*

Description

table_gt_from_ejamit_overall - Create a formatted table of results from EJAM overall summary stats Uses the results_overall element of ejamit() output

Usage

```
table_gt_from_ejamit_overall(
  ejamit_results_1row = NULL,
  type = c("demog", "envt")[1]
)
```

Arguments

`ejamit_results_1row`	1-row data.table like testoutput_ejamit_100pts_1miles$results_overall, as would come from ejamit(testpoints_10)$results_overall
`type`	Must be "demog" or "envt" – Creates one of these at a time

Value

Provides table in gt format from the R package called gt

Examples

```
x <- table_gt_from_ejamit_overall(testoutput_ejamit_100pts_1miles$results_overall)
```

`table_order_variables` *get order of variable names to sort by, as seen in EJScreen Community Report*

Description

get order of variable names to sort by, as seen in EJScreen Community Report

Usage

```
table_order_variables(
  varnames,
  s1 = "ejscreensort",
  s2 = "sortvarlistEJSCREENREPORT",
  s3 = "sort_within_varlistEJSCREENREPORT"
)
```

Arguments

varnames	vector of indicator variables names from blockgroupstats, bgej, etc., such as "pm", "pctlowinc", "pctile.EJ.DISPARITY.traffic.score.eo" etc. and others as found in names_all_r, or specific subsets of those like in c(names_d, names_d_subgroups, names_e) and c(names_ej_pctile, names_ej_state_pctile, names_ej_supp_pctile, names_ej_supp_state_pctile)
s1	name of column in map_headernames to get sort info from
s2	optional like s1 but secondary to s1
s3	optional tertiary

Value

vector as from order(), to be used in sorting a data.frame for example

Examples

```
cbind(table_order_variables(c(names_d, names_d_subgroups, names_e)))

out <- testoutput_ejamit_10pts_1miles
vars <- out$formatted[ , 'indicator']
vars <- fixcolnames(vars, 'long', 'r')
out$formatted[table_order_variables(vars), ]
```

---

table_round — *table_round - round numbers in a table, each column to appropriate number of decimal places*

Description

table_round - round numbers in a table, each column to appropriate number of decimal places

Usage

```
table_round(x, var = names(x), varnametype = "rname", ...)
```

Arguments

x	data.frame, data.table, or vector with at least some numerical columns, like the results of ejamit()$results_bysite
var	optional, but assumed to be names(x) by default, specifies colnames of table or names of vector elements, within x
varnametype	optional, name of column in map_headernames that is looked in for var
...	passed to var_is_numeric_ish()

Value

Returns the original x but with appropriate cells rounded off.

See Also

var_is_numeric_ish() table_rounding_info()

Examples

```
 table_round(c(12.123456, 9, NA ), 'pm')

x <- testoutput_ejamit_10pts_1miles$results_bysite[
  1:2, c('lat','lon', 'pop', names_these, names_ratio_to_avg_these, names_e_pctile),
  with = FALSE
]

table_rounding_info(names(x))

table_round(x)
```

---

table_rounding_info *table_rounding_info - how many decimal places to round to for given variable(s)*

Description

table_rounding_info - how many decimal places to round to for given variable(s)

Usage

```
table_rounding_info(var, varnametype = "rname")
```

Arguments

var	vector of variable names such as c("pctlowinc", "cancer") or c(names_d, names_d_subgroups)
varnametype	which column of map_headernames to use when looking for var, like "rname" or "api" or "long"

Value

named vector same size as var, with var as names.

See Also

varinfo() table_round()

Examples

```
table_rounding_info("pm")
table_round(8.252345, "pm")
table_round(8, "pm")

cbind(table_rounding_info(names_all_r), fixcolnames(names_all_r, "r", "long"))
```

---

table_tall_from_overall

table_tall_from_overall Format the results_overall part of the output of ejamit() or doaggregate()

Description

table_tall_from_overall Format the results_overall part of the output of ejamit() or doaggregate()

Usage

```
table_tall_from_overall(results_overall, longnames)
```

Arguments

results_overall	data.table of 1 row, from output of ejamit() or doaggregate()
longnames	vector of names of variables in results_overall, from output of ejamit() or doaggregate()

Value

data.table that is one row per indicator

Examples

```
table_tall_from_overall(testoutput_ejamit_10pts_1miles$results_overall)
table_tall_from_overall(x$results_bysite[1, ])
```

table_validated_ejamit_row *table_validated_ejamit_row - Cleans/validates EJAM results for 1 place or overall This is a first step in formatting results in nice tables*

Description

table_validated_ejamit_row - Cleans/validates EJAM results for 1 place or overall This is a first step in formatting results in nice tables

Usage

```
table_validated_ejamit_row(ejamit_results_1row = NULL)
```

Arguments

ejamit_results_1row
: 1-row data.table like testoutput_ejamit_100pts_1miles$results_overall,
as would come from ejamit(testpoints_10)$results_overall
or a single row of testoutput_ejamit_100pts_1miles$results_bysite

Value

Returns the input as a 1-row data.table, indicators etc. in the columns. If not a 1 row table, or colnames are not what is expected, it returns correct structure filled with NA values.

Examples

```
x <- table_validated_ejamit_row(testoutput_ejamit_100pts_1miles$results_bysite[ 1, ])
x <- table_validated_ejamit_row(testoutput_ejamit_100pts_1miles$results_overall)
```

table_xls_format *table_xls_format - Format EJAM tabular outputs for saving as Excel spreadsheet Used by table_xls_from_ejam()*

Description

table_xls_format - Format EJAM tabular outputs for saving as Excel spreadsheet Used by table_xls_from_ejam()

Usage

```
table_xls_format(
  overall,
  eachsite,
  longnames = NULL,
  formatted = NULL,
  bybg = NULL,
  plot_distance_by_group = FALSE,
  summary_plot = NULL,
  plotlatest = FALSE,
  plotfilename = NULL,
  mapadd = FALSE,
  ok2plot = TRUE,
  analysis_title = "EJAM analysis",
  buffer_desc = "Selected Locations",
  radius_or_buffer_in_miles = NULL,
  radius_or_buffer_description =
   "Miles radius of circular buffer (or distance used if buffering around polygons)",
  notes = NULL,
  heatmap_colnames = NULL,
  heatmap_cuts = c(80, 90, 95),
  heatmap_colors = c("yellow", "orange", "red"),
  heatmap2_colnames = NULL,
  heatmap2_cuts = c(1.009, 2, 3),
  heatmap2_colors = c("yellow", "orange", "red"),
  hyperlink_colnames = c("EJScreen Report", "EJScreen Map", "ECHO report"),
  graycolnames = NULL,
  narrowcolnames = NULL,
  graycolor = "gray",
  narrow6 = 6,
  testing = FALSE,
  launchexcel = FALSE,
  saveas = NULL,
  ...
)
```

Arguments

overall	table to save in one tab, from ejamit()$overall, EJAM analysis of indicators overall (one row), but if entire output of ejamit() is passed as if it were overall, function figures out eachsite, etc.
eachsite	table to save in one tab, from ejamit()$overall, EJAM analysis site by site (one row per site)
longnames	vector of indicator names to display in Excel table
formatted	optional table to save in one tab, from ejamit()$overall, EJAM analysis overall in different format
bybg	Optional large table of details of each block group that is only needed to analyze distances by group.
plot_distance_by_group	logical, whether to try to add a plot of mean distance by group. This requires that bybg be provided as a parameter input to this function.

summary_plot	optional plot object passed from EJAM shiny app to save in 'Plot' sheet of Excel table
plotlatest	optional logical. If TRUE, the most recently displayed plot (prior to this function being called) will be inserted into a tab called plot2
plotfilename	the full path including name of .png file to insert
mapadd	logical optional - try to include a map of the points
ok2plot	can set to FALSE to prevent plots from being attempted, while debugging
analysis_title	optional title passed from Shiny app to 'Notes' sheet
buffer_desc	optional description of buffer used in analysis, passed to 'Notes' sheet
radius_or_buffer_description	optional text saying if distance is radius or polygon buffer, passed to 'Notes' sheet
notes	Text of additional notes to put in the notes tab, optional vector of character elements pasted in as one line each.
heatmap_colnames	optional vector of colnames to apply heatmap colors
heatmap_cuts	vector of values to separate heatmap colors, between 0-100
heatmap_colors	vector of color names for heatmap bins, same length as heatmap_cuts, where first color is for those >= 1st cutpoint, but <2d, second color is for those >=2d cutpoint but <3d, etc.
hyperlink_colnames	names of which to treat as URLs that should be hyperlinks
graycolnames	which columns to deemphasize
narrowcolnames	which column numbers to make narrow
graycolor	color used to deemphasize some columns
narrow6	how narrow
testing	optional for testing only
launchexcel	Set to TRUE to have this function launch Excel immediately, showing the final workbook created here.
saveas	If not NULL, and a valid path with filename.xlsx is provided, the workbook will be saved locally at that path and name. Warning: it will overwrite an existing file.
...	other params passed along to openxlsx::writeData()
radius_miles	If provided, miles buffer distance (from polygon or from point if circular buffers)

Details

Already took and put here most or all of code from table_xls_format() or table_xls_format_api()

Value

a workbook, ready to be saved in spreadsheet format, with tabs like "Overall" and "Each Site"

See Also

table_xls_from_ejam()

Examples

```
## Not run:
  table_xls_format(
    testoutput_ejamit_100pts_1miles$results_overall,
    testoutput_ejamit_100pts_1miles$results_bysite,
    saveas =  "out1.xlsx")
 # can just pass the whole results of ejamit(), for convenience
 wb <- table_xls_format(testoutput_ejamit_100pts_1miles)
 openxlsx::saveWorkbook(wb, file = "out2.xlsx")

## End(Not run)
```

`table_xls_from_ejam` *table_xls_from_ejam Format the results of ejamit() for excel and optionally save .xlsx file Uses table_xls_format()*

Description

table_xls_from_ejam Format the results of ejamit() for excel and optionally save .xlsx file Uses table_xls_format()

Usage

```
table_xls_from_ejam(
  ejamitout,
  fname = NULL,
  save_now = TRUE,
  overwrite = TRUE,
  launchexcel = FALSE,
  interactive_console = TRUE,
  ok2plot = TRUE,
  in.testing = FALSE,
  in.analysis_title = "EJAM analysis",
  react.v1_summary_plot = NULL,
  radius_or_buffer_in_miles = NULL,
  buffer_desc = "Selected Locations",
  radius_or_buffer_description =
   "Miles radius of circular buffer (or distance used if buffering around polygons)",
  hyperlink_colnames = c("EJScreen Report", "EJScreen Map", "ECHO report"),
  ...
)
```

Arguments

`ejamitout`	output of `ejamit()`
`fname`	optional name or full path and name of file to save locally, like "out.xlsx"
`save_now`	optional logical, whether to save as a .xlsx file locally or just return workbook object that can later be written to .xlsx file using `openxlsx::saveWorkbook()`
`overwrite`	optional logical, passed to `openxlsx::saveWorkbook()`
`launchexcel`	optional logical, passed to `table_xls_format()`, whether to launch browser to see spreadsheet immediately

interactive_console
: optional - should set to FALSE when used in code or server. If TRUE, prompts RStudio user interactively asking where to save the downloaded file

ok2plot
: optional logical, passed to table_xls_format(), whether safe to try and plot or set FALSE if debugging plot problems

in.testing
: optional logical

in.analysis_title
: optional title as character string

react.v1_summary_plot
: optional - a plot object

radius_or_buffer_in_miles
: optional radius in miles

radius_or_buffer_description
: optional text phrase describing places analyzed

hyperlink_colnames
: optional names of columns with URLs

...
: optional additional parameters passed to table_xls_format(), such as heatmap_colnames, heatmap_cuts, heatmap_colors, etc.

Value

returns a workbook object for use by openxlsx::saveWorkbook(wb_out, pathname) or returns just the full path/file name of where it was saved if save_now = TRUE

Examples

```
## Not run:
  table_xls_from_ejam(testoutput_ejamit_10pts_1miles)

## End(Not run)
 @seealso [table_xls_format()]
```

---

testoutput_doaggregate_1000pts_1miles

test output of doaggregate()

Description

test output of doaggregate()

Details

This is the output of doaggregate(testoutput_getblocksnearby_1000pts_1miles, sites2states_or_latlon = testpoints_1000, radius = 1, include_ejindexes = TRUE)

See Also

doaggregate() ejamit() testoutput_getblocksnearby_1000pts_1miles testpoints_1000

testoutput_doaggregate_100pts_1miles

test output of doaggregate()

Description

test output of doaggregate()

Details

This is the output of doaggregate(testoutput_getblocksnearby_100pts_1miles, sites2states_or_latlon = testpoints_100, radius = 1, include_ejindexes = TRUE)

See Also

`doaggregate()` `ejamit()` testoutput_getblocksnearby_100pts_1miles testpoints_100

testoutput_doaggregate_10pts_1miles

test output of doaggregate()

Description

test output of doaggregate()

Details

This is the output of doaggregate(testoutput_getblocksnearby_10pts_1miles, sites2states_or_latlon = testpoints_10, radius = 1, include_ejindexes = TRUE)

See Also

`doaggregate()` `ejamit()` testoutput_getblocksnearby_10pts_1miles testpoints_10

testoutput_ejamit_1000pts_1miles

test output of ejamit()

Description

test output of ejamit()

Details

This is the output of ejamit(testpoints_1000, radius = 1, include_ejindexes = TRUE)

See Also

`doaggregate()` `ejamit()` testoutput_doaggregate_1000pts_1miles and testpoints_1000

`testoutput_ejamit_100pts_1miles`
test output of ejamit()

Description

test output of ejamit()

Details

This is the output of ejamit(testpoints_100, radius = 1, include_ejindexes = TRUE)

See Also

`doaggregate()` `ejamit()` testoutput_doaggregate_100pts_1miles and testpoints_100

`testoutput_ejamit_10pts_1miles`
test output of ejamit()

Description

test output of ejamit()

Details

This is the output of ejamit(testpoints_10, radius = 1, include_ejindexes = TRUE)

See Also

`doaggregate()` `ejamit()` testoutput_doaggregate_10pts_1miles and testpoints_10

`testoutput_getblocksnearby_1000pts_1miles`
test output of getblocksnearby(), and is an input to doaggregate()

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_1000, radius = 1)

See Also

`getblocksnearby()` `doaggregate()` testpoints_1000

`testoutput_getblocksnearby_100pts_1miles` *test output of getblocksnearby(), and is an input to doaggregate()*

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_100, radius = 1)

See Also

`getblocksnearby()` `doaggregate()` testpoints_100

`testoutput_getblocksnearby_10pts_1miles` *test output of getblocksnearby(), and is an input to doaggregate()*

Description

test output of getblocksnearby(), and is an input to doaggregate()

Details

This is the output of getblocksnearby(testpoints_10, radius = 1)

See Also

`getblocksnearby()` `doaggregate()` testpoints_10

`testpoints_10` *test points data.frame with columns siteid, lat, lon*

Description

test points data.frame with columns siteid, lat, lon

test points data.frame with columns siteid, lat, lon

`testpoints_100` *test points data.frame with columns siteid, lat, lon*

Description

test points data.frame with columns siteid, lat, lon

testpoints_1000 *test points data.frame with columns siteid, lat, lon*

Description

test points data.frame with columns siteid, lat, lon

testpoints_10000 *test points data.frame with columns siteid, lat, lon*

Description

test points data.frame with columns siteid, lat, lon

testpoints_n *testpoints_n - Random points in USA - average resident, facility, BG, block, or square mile Get data.table of Random Points (lat lon) for Testing/ Benchmarking/ Demos, weighted in various ways. The weighting can be specified so that each point reflects the average EPA-regulated facility, blockgroup, block, place on the map, or US resident.*

Description

testpoints_n - Random points in USA - average resident, facility, BG, block, or square mile Get data.table of Random Points (lat lon) for Testing/ Benchmarking/ Demos, weighted in various ways. The weighting can be specified so that each point reflects the average EPA-regulated facility, blockgroup, block, place on the map, or US resident.

Usage

```
testpoints_n(
  n = 10,
  weighting = c("frs", "pop", "area", "bg", "block"),
  dt = TRUE,
  ST_needed = NULL
)
```

Arguments

n	Number of points needed (sample size)
weighting	word indicating how to weight the random points (some synonyms are allowed, in addition to those shown here): Note the default is frs, but you may want to use pop even though it is slower. • pop or people = Average Person: random person among all US residents (block point of residence per 2020 Census) • frs or facility = Average Facility: random EPA-regulated facility from actives in Facility Registry Service (FRS)

- bg = Average Blockgroup: random US Census block group (internal point like a centroid)
- block = Average Block: random US Census block (internal point like a centroid)
- area or place = Average Place: random point on a map (internal point of avg blockgroup weighted by its square meters size)

dt — logical, whether to return a data.table (DEFAULT) instead of normal data.frame

ST_needed — optional, can be a character vector of 2 letter State abbreviations to pick from only some States.

Value

data.frame or data.table with columns lat, lon in decimal degrees, and any other columns that are in the table used (based on weighting)

Examples

```
## Not run:
mapfast(testpoints_n(300, ST_needed = c('LA','MS')) )
n=2
for (d in c(TRUE,FALSE)) {
  for (w in c('frs', 'pop', 'area', 'bg', 'block')) {
    cat("n=",n,"  weighting=",w, "  dt=",d,"\n\n")
    print(x <- testpoints_n(n,w,d)); print(class(x))
    cat('\n')
  }
}

## End(Not run)
```

---

test_regid *test_regid (DATA) test data, vector of EPA FRS Registry ID numbers*

Description

test_regid (DATA) test data, vector of EPA FRS Registry ID numbers

Details

For testing, e.g.,

frs_from_siteid(test_regid)

mapfast(frs_from_regid(test_regid))

`trilaterate_sites2blocks` *trilaterate_sites2blocks - Estimate lat,lon of each siteid, from outputs of getblocksnearby() get data.table with siteid, lat,lon of each site (eg for when you did not save sitepoints info)*

Description

trilaterate_sites2blocks - Estimate lat,lon of each siteid, from outputs of getblocksnearby() get data.table with siteid, lat,lon of each site (eg for when you did not save sitepoints info)

Usage

```
trilaterate_sites2blocks(s2b)
```

Arguments

s2b	like testoutput_getblocksnearby_10pts_1miles

Value

a data.table with one row per unique siteid from input dt, plus lat,lon columns

Examples

```
s2b = copy(testoutput_getblocksnearby_10pts_1miles)
s2b_located = latlon_join_on_blockid(s2b) # done by trilaterate also
inferred_sites = trilaterate_sites2blocks(s2b)
inferred_sites
plotblocksnearby(s2b_located)
```

`unshared` *unshared aka setdiff2 - UTILITY - see what is only in x or y but not both utility just like setdiff except for y,x and also x,y Just shows which elements are in one and only one of the sets x and y*

Description

unshared aka setdiff2 - UTILITY - see what is only in x or y but not both utility just like setdiff except for y,x and also x,y Just shows which elements are in one and only one of the sets x and y

Usage

```
unshared(x, y)
```

url_4table *url_4table - Create URLs in columns*

Description

url_4table - Create URLs in columns

Usage

```
url_4table(lat, lon, radius, regid = NULL, as_html = TRUE)
```

Arguments

lat	vector of latitudes
lon	vector of longitudes
radius	vector of values for radius in miles
regid	optional vector of FRS registry IDs if available to use to create links to detailed ECHO facility reports
as_html	logical, optional. passed to url_ejscreen_report() and url_ejscreenmap() from EJAMejscreenapi package

Value

list of data.frames to append to the list of data.frames created by ejamit() or doaggregate(), list(results_bysite = results_bysite, results_overall = results_overall, newcolnames=newcolnames)

See Also

url_ejscreen_report() url_ejscreenmap() url_echo_facility_webpage() from EJAMejscreenapi package

url_bookmark_save *url_bookmark_save save bookmarked EJScreen session (map location and indicator)*

Description

url_bookmark_save save bookmarked EJScreen session (map location and indicator)

Usage

```
url_bookmark_save(..., file = "ejscreenbookmark.json")
```

Arguments

...	passed to url_bookmark_text()
file	path and name of .json file you want to save locally

Details

WORK IN PROGRESS - NOT USED AS OF EARLY 2023. You can use this function to create and save a json file that is a bookmark for a specific place/ map view/ data layer in EJScreen. You can later pull up that exact map in EJScreen by launching EJScreen, clicking Tools, Save Session, Load from File.

***Units are not lat lon: "spatialReference":"latestWkid":3857,"wkid":102100

Note: (1) The number of sessions that can be saved depends on the browser cache size. (2) Session files, if saved, are available from the default Downloads folder on your computer. (3) Users should exercise caution when saving sessions that may contain sensitive or confidential data.

Value

URL for 1 bookmarked EJScreen map location and variable displayed on map

---

`url_bookmark_text` *url_bookmark_text URL for 1 bookmarked EJScreen session (map location and indicator)*

Description

url_bookmark_text URL for 1 bookmarked EJScreen session (map location and indicator)

Usage

```
url_bookmark_text(
  x = c(-13232599.1784247, -13085305.0249191),
  y = c(3970069.24597194, 4067373.582979),
  name = "BookmarkedEJScreenMap",
  title = "Socioeconomic Indicators",
  renderField = "B_UNEMPPCT",
  pctlevel = "nation",
  xmin = 1.1 * min(x),
  xmax = 0.9 * min(x),
  ymin = 0.9 * min(y),
  ymax = 1.1 * min(y),
  urlrest = paste0("https://geopub.epa.gov/arcgis/rest/services",
    "/ejscreen/ejscreen_v2022_with_AS_CNMI_GU_VI/MapServer")
)
```

Arguments

`x`	vector of approx topleft, bottomright longitudes in some units EJScreen uses? Units are not lat lon: "spatialReference":"latestWkid":3857,"wkid":102100
`y`	vector of approx topleft, bottomright latitudes in some units EJScreen uses? Units are not lat lon: "spatialReference":"latestWkid":3857,"wkid":102100
`name`	Your name for the map bookmark
`title`	Your name for the map like Socioeconomic Indicators or Pollution and Sources
`renderField`	name of variable shown on map, like B_UNEMPPCT for map color bins of percent unemployed or B_PTRAF for traffic indicator

pctlevel	nation or state
xmin	calculated bounding box for map view
xmax	calculated bounding box for map view
ymin	calculated bounding box for map view
ymax	calculated bounding box for map view
urlrest	Just use the default but it changes each year

Details

WORK IN PROGRESS - NOT USED AS OF EARLY 2023. You can use this function to create and save a json file that is a bookmark for a specific place/ map view/ data layer in EJScreen. You can later pull up that exact map in EJScreen by launching EJScreen, clicking Tools, Save Session, Load from File.

Note: (1) The number of sessions that can be saved depends on the browser cache size. (2) Session files, if saved, are available from the default Downloads folder on your computer. (3) Users should exercise caution when saving sessions that may contain sensitive or confidential data.

Value

URL for 1 bookmarked EJScreen map location and variable displayed on map

See Also

url_bookmark_save()

Examples

```
## Not run:
  url_bookmark_text()
  url_bookmark_save(
    x=c(-10173158.179197036, -10128824.702791695),
    y=c(3548990.034736070,3579297.316451102),
    file="./mysavedejscreensession1.json")

## End(Not run)
```

---

url_countyhealthrankings

url_countyhealthrankings

Description

url_countyhealthrankings

Usage

```
url_countyhealthrankings(fips, year = 2023)
```

Arguments

fips	vector of fips codes of counties, 5 characters each, like "10003"
year	2023

Value

vector of URLs

`url_getacs_epaquery` *url_getacs_epaquery - experimental/ work in progress: get ACS data via EPA API (for <200 places)*

Description

uses ACS2019 rest services ejscreen ejquery MapServer 7

Documentation of format and examples of input parameters:

https://geopub.epa.gov/arcgis/sdk/rest/index.html#/Query_Map_Service_Layer/02ss0000000r000000/

Usage

```
url_getacs_epaquery(
  objectIds = 1:3,
  servicenumber = 7,
  outFields = NULL,
  returnGeometry = FALSE,
  justurl = FALSE,
  ...
)
```

Arguments

`objectIds`	see API
`servicenumber`	see API
`outFields`	see API. eg "STCNTRBG","TOTALPOP","PCT_HISP",
`returnGeometry`	see API
`justurl`	if TRUE, returns url instead of default making API request
`...`	passed to url_getacs_epaquery_chunked()

Value

table

Examples

```
url_getacs_epaquery(justurl=TRUE)
```

url_getacs_epaquery_chunked *url_getacs_epaquery_chunked - experimental/ work in progress: in chunks, get ACS data via EPA API*

Description

url_getacs_epaquery_chunked - experimental/ work in progress: in chunks, get ACS data via EPA API

Usage

```
url_getacs_epaquery_chunked(
  objectIds = 1:3,
  servicenumber = 7,
  outFields = NULL,
  returnGeometry = FALSE,
  justurl = FALSE,
  chunksize = 200,
  ...
)
```

Arguments

objectIds	see API
servicenumber	see API
outFields	see API
returnGeometry	see API
justurl	see API
chunksize	eg 200 for chunks of 200 each request
...	passed to url_getacs_epaquery()

Value

table

Examples

```
## Not run:
 # x <- list() # chunked chunks. best not to ask for all these:
 # x[[1]] <- url_getacs_epaquery_chunked(   1:1000, chunksize = 100)
 # x[[2]] <- url_getacs_epaquery_chunked(1001:5000, chunksize = 100)
 # xall <- do.call(rbind, x)

## End(Not run)
```

`url_get_eparest_chunked_by_id` *url_get_eparest_chunked_by_id - experimental/ work in progress: in chunks, get ACS data or Block weights nearby via EPA API*

Description

url_get_eparest_chunked_by_id - experimental/ work in progress: in chunks, get ACS data or Block weights nearby via EPA API

Usage

```
url_get_eparest_chunked_by_id(objectIds, chunksize = 200, ...)
```

Arguments

`objectIds`	see API
`chunksize`	see API
`...`	passed to url_getacs_epaquery()

Value

a table

`url_get_via_url` *url_get_via_url - helper function work in progress: GET json via url of ejscreen ejquery map services*

Description

url_get_via_url - helper function work in progress: GET json via url of ejscreen ejquery map services

Usage

```
url_get_via_url(url)
```

Arguments

`url`	the url for an EJScreen ejquery request

Value

json

`url_naics.com` *url_naics.com - Get URL for page with info about industry sectors by text query term See (https://naics.com) for more information on NAICS codes*

Description

url_naics.com - Get URL for page with info about industry sectors by text query term See (https://naics.com) for more information on NAICS codes

Usage

```
url_naics.com(query, as_html = FALSE, linktext)
```

Arguments

`query`	string query term like "gasoline" or "copper smelting"
`as_html`	Whether to return as just the urls or as html hyperlinks to use in a DT::datatable() for example
`linktext`	used as text for hyperlinks, if supplied and as_html=TRUE

Value

URL as string

`usastats` *usastats (DATA) data.frame of 100 percentiles and means*

Description

data.frame of 100 percentiles and means (about 100 rows) in the USA overall, across all locations (e.g., block groups in blockgroupstats) for a set of indicators such as percent low income. Each column is one indicator (or specifies the percentile).

This should be similar to the lookup tables in the gdb on the FTP site of EJScreen, except it also has data for the demographic race/ethnicity subgroups. For details on how the table was made, see /EJAM/data-raw/usastats_subgroups.R

usastats_means — *usastats_means - convenient way to see US MEANS of ENVIRONMENTAL and DEMOGRAPHIC indicators*

Description

usastats_means - convenient way to see US MEANS of ENVIRONMENTAL and DEMOGRAPHIC indicators

Usage

```
usastats_means(
  varnames = c(EJAM::names_e, EJAM::names_d, EJAM::names_d_subgroups_nh),
  PCTILES = NULL,
  dig = 2
)
```

Arguments

varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	how many digits to round to

usastats_query — *usastats_query - convenient way to see US mean, pctiles of Envt and Demog indicators in lookup table*

Description

usastats_query - convenient way to see US mean, pctiles of Envt and Demog indicators in lookup table

Usage

```
usastats_query(
  varnames = c(EJAM::names_e, EJAM::names_d, EJAM::names_d_subgroups_nh),
  PCTILES = NULL,
  dig = 2
)
```

Arguments

varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	how many digits to round to

Details

A long list of variables: usastats_query(intersect(EJAM::names_all_r, names(EJAM::usastats)))

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us                # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector

usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))

## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')

## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),  .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
```

```
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

---

usastats_queryd *usastats_queryd - convenient way to see US mean, pctiles of DEMOGRAPHIC indicators in lookup table*

Description

usastats_queryd - convenient way to see US mean, pctiles of DEMOGRAPHIC indicators in lookup table

Usage

```
usastats_queryd(
  varnames = c(EJAM::names_d, EJAM::names_d_subgroups_nh),
  PCTILES = NULL,
  dig = 2
)
```

Arguments

varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	how many digits to round to

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us                # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector

usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
```

```
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))

## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')

## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),  .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

`usastats_querye` *usastats_querye - convenient way to see US mean, pctiles of ENVIRONMENTAL indicators in lookup table*

Description

usastats_querye - convenient way to see US mean, pctiles of ENVIRONMENTAL indicators in lookup table

Usage

```
usastats_querye(varnames = EJAM::names_e, PCTILES = NULL, dig = 2)
```

Arguments

varnames	names of columns in lookup table, like "proximity.rmp"
PCTILES	vector of percentiles 0-100 and/or "mean"
dig	how many digits to round to

Examples

```
## Not run:

usastats_querye()
#  data.frame where names_e are the names(),
#  means plus other percentiles, and there are other cols REGION PCTILE

avg.in.us                # This is a data.frame, 1 row, where colnames are indicators
avg.in.us[names_e]          # subset is a data.frame!
unlist(avg.in.us[names_e])  # to make it a vector

usastats_means()        # This is a matrix, with 1 col, and indicator names are rownames
usastats_means(names_e)     # subset is a matrix        and indicator names are rownames
usastats_means()[names_e, ] # subset is a named vector  and indicator names are  names

usastats_means()
statestats_query()

statestats_query()[,names_d]
statestats_query(varnames = names_d)

statestats_query()[,names_e]
statestats_query(varnames = names_e)

statestats_query(varnames = names_d_subgroups)
head(statestats_query(varnames = longlist))

## in USA overall, see mean and key percentiles for all demog and envt indicators
usastats_query() # or statestats_query('us') # can say us or US or USA or usa etc.
usastats_query(PCTILES = 'mean')
usastats_means() # same but nicer looking format in console
usastats_means(dig=4)

# long list of variables:
x = intersect(EJAM::names_all_r,  names(EJAM::usastats))
usastats_means(x)

usastats[!(usastats$PCTILE < 50), c("PCTILE", names_d)]
usastats[!(usastats$PCTILE < 50), c("PCTILE", names_e)]

## in 1 state, see mean and key percentiles for all demog and envt indicators
statestats_query('MD')

## in 1 state, see mean and key percentiles for just demog indicators
statestats_queryd('MD')
```

```
## 1 indicator in 1 state, see a few key percentiles and mean
statestats_query('MD','proximity.tsdf')

## mean of 1 indicator for each state
statestats_query(varnames = 'proximity.tsdf')

## using full blockgroup dataset, not lookup tables of percentiles,
blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)), .SDcols= c(names_d, names_e)]

##   see all total counts (not just US means),
##   demographics including subgroups,
##   but not environmental indicators.
t(round(ustotals2(bg = blockgroupstats),2)) #  ustotals2 is from EJAMbatch.summarizer package
t(blockgroupstats[, lapply(.SD, function(x) mean(x, na.rm=T)),
    .SDcols= c(names_e, names_d)])


## End(Not run)
```

`varinfo` *varinfo - Get metadata for a variable, like its type, definition, decimalS rounding, etc. This is just a way to query map_headernames, which has info about each indicator or variable used in EJAM.*

Description

varinfo - Get metadata for a variable, like its type, definition, decimalS rounding, etc. This is just a way to query map_headernames, which has info about each indicator or variable used in EJAM.

Usage

```
varinfo(
  var = map_headernames$rname,
  info = colnames(map_headernames),
  varnametype = "rname"
)
```

Arguments

`var`	vector of variable names such as c("pctlowinc", "cancer") or c(names_d, names_d_subgroups) (and must be found in the column of map_headernames indicated by varnametype parameter below).
`info`	types of metadata/info needed, such as "decimals", "long", etc. which should be among colnames of map_headernames, or alias like "long" as allowed by fixcolnames()
`varnametype`	optional. colname of map_headernames to use when looking for var, like "rname" or "api" or "long"

Details

See map_headernames for what kind of information is available there. But if a variable appears twice+ in var or in map_headernames, info returned only for the 1st row of those

Value

data.frame of 1 or more rows, 1 or more columns, where

rowsnames are var (indicators like "pctmin")

colnames are info (metadata like "decimals")

Cells of table are metadata such as what type of indicator is that var, how many decimal places of rounding should be displayed for it in tables, etc.

Results can be character, numeric, etc. depending on what info is requested

See Also

fixcolnames() table_rounding_info()

Examples

```
varinfo("traffic.score", "decimals")
varinfo(names_d, "long")
myvars <- c(names_d, names_d_subgroups, names_e)
myinfo <- "percentage"
cbind(  is.a.percentage = varinfo(myvars, myinfo) )
cbind(varinfo(names_all_r, "pctile."))
myinfo <- "long"
cbind(varinfo(myvars, myinfo) )
table_rounding_info(names_e)

varinfo(
 var = c(names_these, names_d_pctile),
 info = c(
 "topic_root_term", "varcategory", "vartype", "percentage", "pctile.", "calculation_type"
))

varinfo(names_all_r, c("varcategory", "varlist", "in_api", "in_bgcsv"))
```

---

varname2color_ejam *varname2color_ejam - helper function - for color coding excel sheet columns Convert R variable name of indicator to appropriate color for header row in Excel*

Description

varname2color_ejam - helper function - for color coding excel sheet columns Convert R variable name of indicator to appropriate color for header row in Excel

Usage

```
varname2color_ejam(varname, varnameinfo)
```

Arguments

varname	things like us.avg.pctlowinc

Value

vector of colors

See Also

varname2vartype_ejam() varname2varcategory_ejam() vartype_cat2color_ejam()

---

varname2varcategory_ejam *varname2varcategory_ejam - helper function - given indicator names, look up what category each is*

Description

varname2varcategory_ejam - helper function - given indicator names, look up what category each is

Usage

```
varname2varcategory_ejam(varname, varnameinfo)
```

Arguments

varname	vector of 1 or more names like "pctlowinc" as in unique(map_headernames$rname)
varnameinfo	data.frame with info on type of each variable

Details

tells if variable is "Demographic" "Environmental" "EJ Index" or "other" as from dput(unique(map_headernames$varcate

Value

vector same size as varname

See Also

vartype_cat2color_ejam() varname2color_ejam()

varname2vartype_ejam *varname2vartype_ejam - helper function - given indicator names, look up what type each is*

Description

varname2vartype_ejam - helper function - given indicator names, look up what type each is

Usage

```
varname2vartype_ejam(varname, varnameinfo)
```

Arguments

varname	vector of 1 or more names
varnameinfo	data.frame with info on type of each variable

Details

The types are things like raw data count for indicator, average, percentile, etc.

Value

vector same size as varname

See Also

vartype_cat2color_ejam() varname2color_ejam()

vartype_cat2color_ejam *vartype_cat2color_ejam - helper function - assign fill color to shade excel cells by indicator type and category Use color shading to make spreadsheet easier to use, grouping the indicators*

Description

vartype_cat2color_ejam - helper function - assign fill color to shade excel cells by indicator type and category Use color shading to make spreadsheet easier to use, grouping the indicators

Usage

```
vartype_cat2color_ejam(vartype = raw, varcategory = "other")
```

Arguments

vartype	must be one found in dput(unique(map_headernames$vartype)) like "usratio", "stateratio", "usraw", "stateraw", "uspctile", "statepctile", "usavg", "stateavg", etc. NA if not found.
varcategory	must be one of "Demographic" "Environmental" "EJ Index" "other" as from dput(unique(map_headernames$varcategory))

Value

vector of colors like c('lightblue', 'gray') matching length of vartype

See Also

varname2vartype_ejam() varname2varcategory_ejam() varname2color_ejam()

`var_is_numeric_ish` *var_is_numeric_ish - see which columns seem numeric and could be rounded, e.g. - DRAFT NOT FULLY TESTED*

Description

var_is_numeric_ish - see which columns seem numeric and could be rounded, e.g. - DRAFT NOT FULLY TESTED

Usage

```
var_is_numeric_ish(
  x,
  only.if.already.numeric = FALSE,
  strip.characters.before.coerce = FALSE
)
```

Arguments

`x`	data.table, data.frame, or vector
`only.if.already.numeric`	logical, if TRUE, only reports TRUE for a column (or element) if is.numeric() is TRUE for that one
`strip.characters.before.coerce`	logical, if TRUE, tries to remove spaces and percentage signs before trying to coerce to numeric

Value

logical vector as long as NCOL(x) i.e., is length(x), if x is table, or length(x) if vector

See Also

table_round()

Index