Package ‘neaR’

July 26, 2017

Version 1.0.0

Date 2017-07-25

Title A package providing data processing functions for arts administrative data

Author Greer Mellon[aut]

Maintainer Greer Mellon <`greer.mellon@gmail.com`>

Description
This package provides functions for geo-processing data, creating relevant metrics for analysis, and downloading and appending census data relevant to analysis of U.S. arts data.

Depends R (>= 3.3.2),stringr ,tigris ,RJSONIO,RDSTK ,progress, sp, httr

License What license is it under?

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1.9000

R topics documented:

address2LatLon 2
append_poverty_data 2
create_boolean_urban 3
create_full_address 3
create_intl_flag 4
create_poverty_flag 4
create_urban_type 5
dtf 5
get_cd_data 6
get_county_data 6
get_ct_data 7
get_geocode_data 8
get_msa_data 8
get_padded_zip 9
get_poverty_rates 10
makeChunkIndex 10
tract_level_data 11

Index **12**

address2LatLon *Geocodes data using RDSTK in a loop*

Description

This function loads geocodes data using RDSTK, and operates in loops of chunks set with the chunk index function. It is an intermediary function used in other geocoding functions in this package

Usage

```
address2LatLon(addresses)
```

Arguments

`addresses`	vector of addresses

Value

dataframe of addresses, lat, and long

append_poverty_data *Appends Poverty Data to main Data file*

Description

This function makes it easy to append 2011-2015 poverty rate data from the 5 year ACS to another dataset. To use, just create a new column in the original data to store the poverty rate data, and identify which column in the original data has the census tract ID variables. The function will do the rest.

Usage

```
append_poverty_data(ID_var, pov_column)
```

Arguments

`ID_var`	column in data frame with CT ids, i.e. NEA$CT_GEOID
`pov_column`	the name of the column in the dataset to append poverty data to

Value

A vector of poverty data, in the same order as original dataset

Examples

```
NEA$poverty_rate<-NA
NEA$poverty_rate<-append_poverty_data(NEA$CT_GEOID, NEA$poverty_rate)
```

`create_boolean_urban` *Create Rural/Urban Identifier*

Description

This function can be used to create a column that flags records as either urban or rural.

Usage

```
create_boolean_urban(MSA_category, Latitude)
```

Arguments

`Latitude`	Column/vector with latitude
`MSA_catory`	Name of column with MSA categories - i.e. M1, M2. should be LSAD

Details

Urban records are records that fall into a Metropolitan or Micropolitan statistical area. As a result, MSAs will need to be appended to the data before this function is utilized. Records not falling into these MSAs are classified as rural. Records without valid Lat/Long will result in NAs

Note, caution should be exercised with international records, they are defaulted to "rural" since they will not match a MSA. After adding rural/urban a international record flag should be used to subset the data.

Value

a new column with markers for Rural or Urban

`create_full_address` *Combines relevant columns to create a complete address*

Description

This function combines the address component columns in a dataframe and returns an address vector. Also includes functions to remove problematic characters from the text that will interfere with geocoding

Usage

```
create_full_address(data, address.vars)
```

Arguments

`data`	name of dataset
`address.vars`	vector of address variables i.e. c("Address1", "Address2", "City")

Value

formatted vector of addresses

Examples

```
NEA$full_address<-create_full_address(NEA, c("CoAddress1", "CoAddress2",
"CoCity", "CoState", "CoZip"))
```

---

create_intl_flag *Create Flag for International Records*

Description

This function can be used to create a column that flags intl records. It assumes that the dataset includes a column with State Names

Usage

```
create_intl_flag(state_var, additions = c())
```

Arguments

`state_var`	Name of column in data set with state abbreviations
`additions`	Additional State Names to add as international. defaults to "FO", "AS", "FM", "GU","MH", "MP", "PW", "PR", "VI", "AE","AP", "AA", "CM"

Value

A new Boolean TRUE and FALSE Column where TRUE=International

---

create_poverty_flag *Create Poverty/Not Poverty Flag*

Description

This function can be used to create a column that categorizes records based on their poverty rate.

Usage

```
create_poverty_flag(pov_column, Latitude, cutoff = 20)
```

Arguments

`pov_column`	Name of column with poverty rate - i.e. 20, 30 etc.
`Latitude`	Column/vector with latitude
`cutoff`	cutoff for defining a high poverty neighborhood, defaults to 20

Details

It requires that the dataset already has a column for poverty rate for the census tract in which the record is located, which can be created using the relevant poverty packages in this r package.

Note, that areas without population or without poverty rates are labeled "Missing Data"

Value

a new column with markers for Rural or Urban

`create_urban_type` *Create Categories of Urban Records*

Description

This function can be used to create a column that categorizes records based on urban population size

Usage

```
create_urban_type(MSA_pop, Boolean_Urban_MSA)
```

Arguments

`MSA_catory`	Name of column with MSA categories - i.e. M1, M2. should be LSAD
`Latitude`	Column/vector with latitude

Details

It requires that the dataset already has a rural/urban marker, and that it has a second column with MSA and NECTA population data - should be downloaded and appended from the Census, and matched based on CBSA and NECTA ids.

Note, caution should be exercised with international records, they are defaulted to "rural" since they will not match a MSA. After adding rural/urban a international record flag should be used to subset the data.

Value

a new column with markers for Rural or Urban

`dtf` *Transform to Data Frame*

Description

Converts object to dataframe, defaults to StringsAsFactors=F

Usage

```
dtf(..., StAsFa = FALSE)
```

Value

A data frame

`get_cd_data` *Get Dataframe of Matching Congressional District*

Description

This function returns a dataframe with Congressional data for a vector of Latitudes and Longitudes in a dataframe.

Usage

```
get_cd_data(Latitude, Longitude)
```

Arguments

`Latitude`	vector of Latitudes
`Longitude`	Vector of Longitudes

Details

The package operates by matching Lat/Long to shapefiles of 114th Congressional Districts - valid from 2015-2017.

It returns a dataframe with the following components: GEOID: The GEOID of the CD NAME_LSAD: The Name of the CD

Value

A datafame with associated CD info. Will be in correct order as original Lat/Long to make it easy to append the data

Examples

```
cds<-get_cd_data(NEA$CoLatitude, NEA$CoLongitude)
NEA$CD_GEOID<-cds$CD_GEOID
NEA$CD_NameLSAD<-cds$CD_NameLSAD
```

`get_county_data` *Get Dataframe of Matching County Data*

Description

This function returns a dataframe with County ID data for a vector of Latitudes and Longitudes in a dataframe.

Usage

```
get_county_data(Latitude, Longitude, year = 2015)
```

Arguments

`Latitude`	vector of Latitudes
`Longitude`	Vector of Longitudes
`year`	vintage of shapefiles, defaults to 2015 if none specified

Details

It returns a dataframe with the following components: GEOID: The GEOID of the County - i.e. FIPS Code NAME_LSAD: The Name of the County

Value

A datafame with associated Ccountyinfo. Will be in correct order as original Lat/Long to make it easy to append the data

Examples

```
counties<-get_cunty_data(NEA$CoLatitude, NEA$CoLongitude)
NEA$county_GEOID<-counties$county_GEOID
NEA$county_NameLSAD<-counties$county_NameLSAD
```

`get_ct_data` *Get Dataframe of Matching Census Tract IDs*

Description

This function returns a dataframe with Census Tract ID data for a vector of Latitudes and Longitudes in a dataframe.

Usage

```
get_ct_data(Latitude, Longitude)
```

Arguments

`Latitude`	vector of Latitudes
`Longitude`	Vector of Longitudes

Details

Census Tract shapefiles can only be downloaded at the state level; as such, this function runs in a loop, downloading and matching the shapefile for each state. All shapefiles are the most precise boundary option with the exception of CA, which offers only smaller boundary data

It returns a dataframe with the following components: CT_GEOID: The GEOID of the Census Tract CT_NAME_LSAD: The Name of the Census Tract

It is IMPORTANT to note that the CT GEOID is an 11 character digit It is recommended that CT are matched at the same time that CT level data will be matched. If the file is saved and reimported, the ID must be padded to 11 digits, and must be set to not read out as scientific notation.

Value

A datafame with associated CT info. Will be in correct order as original Lat/Long to make it easy to append the data

Examples

```
CT<-get_ct_data(NEA$CoLatitude, NEA$CoLongitude)
NEA$CT_GEOID<-CT$CT_GEOID
NEA$CT_NameLSAD<-CT$CT_NameLSAD
```

`get_geocode_data` *Primary Geocoding Function*

Description

This function takes an address column that has been properly formatted using get_padded_zip and then create_full_address and returns a dataframe of addresses and relevant lat/long

Usage

```
get_geocode_data(address)
```

Arguments

`address`	name of the address vector in the data set i.e. NEA$full_address

Value

dataframe of addresses and geocodes, that can then be added to the original data

Examples

```
addresses<-get_geocode_data(NEA$full_address)
NEA$CoLongitude<-addresses$long
NEA$CoLatitude<-addresses$lat
```

`get_msa_data` *Get Dataframe of Matching Metropolitan Area Codes*

Description

This function returns a dataframe with MSA data for a vector of Latitudes and Longitudes in a dataframe.

Usage

```
get_msa_data(Latitude, Longitude, year = 2016)
```

Arguments

`Latitude`	vector of Latitudes
`Longitude`	Vector of Longitudes

Details

The package operates by matching Lat/Long to shapefiles of Metropolitan and Micropolitan Statistical Areas - from CBSA files. Then, the package matches Lat/Long that have not yet been coded to Metropolitan and Micropolitan NECTAS

It returns a dataframe with the following components: GEOID: The GEOID of the MSA or NECTA NAME_LSAD: The Name of the MSA or NECTA LSAD: A Code indicating whether it is a Metro or Micro MSA or NECTA -M1 are Metropolitan MSAs -M2 are Micropolitan MSAs -M5 are Metropolitan NECTAS -M6 are Micropolitan NECTAS Source: A constructed variable, indicating if the boundaries were based on NECTA or MSAs

Value

A datafame with associated MSA info. Will be in correct order as original Lat/Long to make it easy to append the data

Examples

```
msas<-get_msa_data(NEA$CoLatitude, NEA$CoLongitude)
NEA$MSA_GEOID<-msa$csba_GEOID
NEA$MSA_NameLSAD<-msa$cbsa_NameLSAD
```

`get_padded_zip` *Pads zipcodes to 5 digits*

Description

This function uses the stringr package to automate padding zipcodes to 5 digits

Usage

```
get_padded_zip(zip)
```

Arguments

`zip`	vector of zipcodes, five digit format

Value

vector of padded zipcodes

Examples

```
NEA$zip<-get_padded_zip(NEA$CoZip)
```

`get_poverty_rates`	*Creates a poverty rate variable, and merges two files with different census tables*

Description

This function requires two datasets using the tract level data function: -a dataframe that has the poverty table data "B17021_002E" -a second dataframe that has the population data for which poverty is known "B17021_001E"

Usage

```
get_poverty_rates(poverty, population)
```

Arguments

`poverty`	dataframe with poverty table
`population`	dataframe with population table

Value

A dataframe with formatted/merged pop and poverty data

`makeChunkIndex`	*Create chunks for looped functions*

Description

This function loads a file as a matrix. It assumes that the first column contains the rownames and the subsequent columns are the sample identifiers. Any rows with duplicated row names will be dropped with the first one being kepted.

Usage

```
makeChunkIndex(x, group.size)
```

Arguments

`x`	vector to be used in function
`group.size`	Sets the size of the "chunk" to be run at one time

Value

Chunk Index

`tract_level_data` *Download CT Level data from Census API*

Description

This function downloads a national-level file of tract-level data from a specified census table. Data is downloaded in a loop

Usage

```
tract_level_data(year = "2015", survey = "acs5", table = "B17021_002E")
```

Arguments

`year`	ending year of ACS data, defaults to 2015
`survey`	survey used, defaults to acs5
`table`	table used, defaults to "B17021_002E"

Value

A dataframe with census data

Index

address2LatLon, 2
append_poverty_data, 2

create_boolean_urban, 3
create_full_address, 3
create_intl_flag, 4
create_poverty_flag, 4
create_urban_type, 5

dtf, 5

get_cd_data, 6
get_county_data, 6
get_ct_data, 7
get_geocode_data, 8
get_msa_data, 8
get_padded_zip, 9
get_poverty_rates, 10

makeChunkIndex, 10

tract_level_data, 11