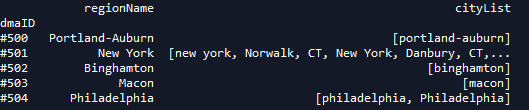
DMAmapper Documentation

This module is intended to facilitate the organization and conversion of DMA number, official Region names, and associated cities/aliases of regions. The code works by utilizing a pandas dataframe for data management, storing the dataframe in a .csv located in the same folder, and reading back from that .csv as needed. The .csv is generated by the program and is called ‘DMA Dataframe.csv’. The program can be invoked on its own by editing the *main()* function in DMAmapper.py and running the code. It can also be imported into another module by including the line**: import DMAmapper as dm**. The dataframe is defined with unique dma numbers as indices and two columns: one is the official region name for the market and the other is a list of cities/subregions contained in the region.



The primary function inside the module is called **ReturnInfo()**. By passing this function the dataframe and one of three pieces of information, it returns the other two in a list. The function looks like this:

def **ReturnInfo**(inDF, dma = None, region = None, city = None):

* ‘inDF’ is the main dataframe and does not need any keyword arguments.
* 'dma' is the unique number that defines that regional market.
* 'region' is the official name of the regional market.
* 'city' can be an alias for the market, a city contained in the market, or a submarket.

Example function calls and output:

*Function call*: print(ReturnInfo(dmaDF, dma = '#501'))

*Output*: ['New York', ['new york', 'Norwalk, CT', 'New York', 'Danbury, CT', 'Newark, NJ', 'New Brunswick, NJ']]

*Function call*: print(ReturnInfo(dmaDF, region = 'New York'))

*Output*: ['#501', ['new york', 'Norwalk, CT', 'New York', 'Danbury, CT', 'Newark, NJ', 'New Brunswick, NJ']]

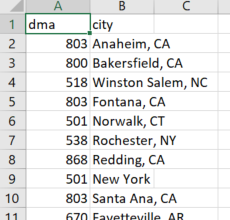
*Function call*: print(ReturnInfo(dmaDF, city = 'Danbury, CT'))

*Output*: [['#501', 'New York']

*Note*: The dma argument requires a valid 3 digit number with a preceding hashtag to work correctly. Also, at this time the function cannot accept more than one optional argument.

There are two secondary functions included in the module: **AddNewCities()** and **MissingCities()**.

**AddNewCities()** can add a properly formatted .csv of cities and subregions to the dataframe and returns the updated dataframe. The uploaded file MUST contain only two columns, the first of which is dma numbers (which must be valid, 3-digit integers) and the second of which must be the name of the city to be added. The function assumes a header on the first row. See example below.



def **AddNewCities**(inFile, topDF):

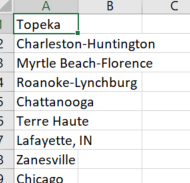
* ‘inFile’ is the name of the csv file to be uploaded.
* ‘topDF’ is the main dataframe passed into the function.

Example function call and output:

*Function call:* dmaDF = AddNewCities('example.csv', dmaDF)

*Output:* -None-

**MissingCities()** can accept a csv file containing city and subregion names and returns a list of those not currently found under any region. It does not add these cities to the database but tells the user which cities do need to be tagged with their dma and uploaded via the AddNewCities() function. An example of the csv file is below.



def **MissingCities**(inDF, filename):

* ‘inDF’ is the main dataframe passed into the function.
* ‘filename’ is the name of the csv file to be checked.

Example function call and output:

*Function call*: print(MissingCities(dmaDF, 'example cities.csv'))

*Output*: [‘Washington D.C.’, ‘Austin, TX’]