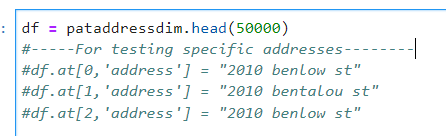
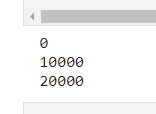
**Instructions for running:**

The input file that has been used for all the geocoding is Pataddressdim.csv and unless that changes, the file name/path does not need to be changed. For testing purposes, the amount of cells that are geocoded can be changed by altering the number in head or getting rid of it altogether to run the full set of addresses. The commented-out lines following this line can be used to test specific addresses to see how the script standardizes/groups them. (Since the first 50 addresses or so have the same eMRN, addresses that are similar enough will be grouped) Tail or other row slices can’t be used do to indexing mismatches



As you run the code, it will print out the cell number every 10,000 rows as a marker of the codes progress:



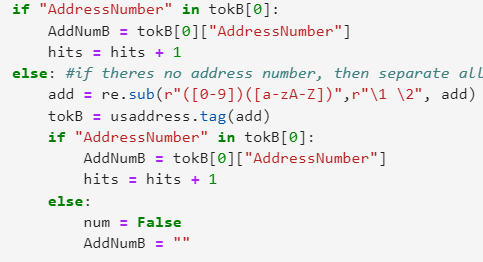
**Summary**

Cell 1: Imports

Cell 2: Open file and reassigning column names

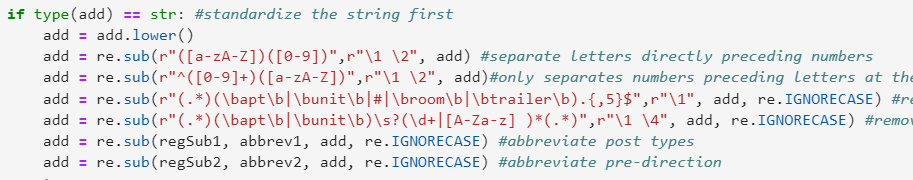
Cell 3: Regular expression strings and Functions:

* The regex strings contain the keywords that will be abbreviated to make the addresses standardized and easier to group together
* Functions for abbreviating parts in an address: 1st is the address post type (e.g. street to st.) and 2nd is the pre-direction (e.g. north to n.)
* The **parse** function accepts the tokenized dictionary returned by running the address on the usaddress tag function. It then scans through the parts and determines if any components (street-name, address number, post-type, and pre-direction) are missing. If so, it will return each component and a new address with those parts put together. If not, the addresses will be modified to see if that will allow the usaddress to properly split the address into components. Each of the above components will be returned, along with the combined address, and finally the total number of components is returned as well



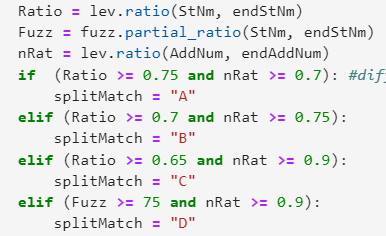
In this example, the AddressNumber component is checked and if its not there, then re.sub is run to separate numbers from letters anywhere in the address in the case that the number is being grouped with the street name. The usaddress.tag is run on add again to see if there is a hit

* The **processAdd** function is run before the parse function and is run on every function. It removes any apt, unit, or room tags at the end of the address and separates any numbers connected to letters, and abbreviates post-type/prefixes.



* It then tries to tokenize the addresses in a way that does not produce duplicate components (the try and except are for catching duplicate tag errors). If there are duplicates, there likely is a repeat of the address so anything after the post-type of the address, e.g. street, is cut off. This is first done to full post types, and if this still fails, then this is done with abbreviated post-types. The parse function is then run on the tokenized address.

Cell 4:

* First a copy of the dataframe is made so that it can be edited without modifying the original. Then the address and address components are obtained by running processAdd and parse on the first address in the dataframe and are set to the set of variables named endAdd, endAddNum, etc.
* **Loop**:
  + Next for each address in the dataframe, the address is processed and then each of the components are obtained and named rowAdd, rowAddNum etc. and these components are compared to the endAdd components to see if they match. If the e\_mrn matches as well, then the two rows are combined and one of them is deleted. The address with the greater number of component hits (indicative of a more complete address) is the one taken to represent the group.
    - The comparison process involves taking the levenshtein distance between the two address street names (number of insertions/deletions compared to how long the string is) to determine a ratio of how similar the two strings are and doing the same for the address numbers and seeing if it meets the tested cutoffs. There are three different levels of cutoffs, depending on the similarity of the street name and the similarity of the address numbers. An alternative check we have is for partial-levenshtein matches (where substrings would count as complete matches) for cases where street names are duplicated or excess information is included. (e.g. 5 northern northern rd would match 5 northern rd) Since this method is more lenient, the ratio for the address number needed is greater.
    - 

The ratios for the 3 lev matches and the fuzzy partial match

* + - If none of these 4 methods yields a match (perhaps because it was unable to be split), then the full addresses themselves are compared
  + If the two addresses don’t match, indicative of a completely new address, then the decided upon final address and its relevant details are saved into the new dataframe (df2) and this process is repeated by setting the endAdd equal to the next address.

FIPS code:

Cell 1: imports

Cell 2: Open file (SAS output file) and reassign column names

Cell 3: functions

* **stateNum** converts the state abbreviation to its 2-digit code
* **countyNum** and **tractNum** add preceding zeros to the code until they reach the desired length
* **convert** turns zip code floats into string

Cell 3: main loop

* loop through all the rows of addresses and note if the original address and new address are both street matched and check if the FIPs code of each address matches. If not, then check which parts of the addresses (state, county, tract, and bloc) do not match