DBF to CSV function

Here is a simple function that can convert a single dbf file to csv format.

Function makes use of the dbf read module which will input a dbf and output a csv file in the same location as the DBF files.

Please note that not all DBF files can be read in Excel or the Statistical R Package, this the birth of this function.

This function can also be mapped to several files in an interation.

Simple process ¶

import csv

from dbfread import DBF

Input a dbf, output a csv, same name, same path, except extension

Set the csv file name

Table variable is a DBF object

Create a csv file, fill it with dbf content

Write the column name

Return the csv name and file

In [5]:
```
import csv
from dbfread import DBF

def dbf_to_csv(dbf_table_pth):#Input a dbf, output a csv, same name, same p
ath, except extension
    csv_fn = dbf_table_pth[:-4]+ ".csv" #Set the csv file name
    table = DBF(dbf_table_pth)# table variable is a DBF object
    with open(csv_fn, 'w', newline = '') as f:# create a csv file, fill it
 with dbf content
        writer = csv.writer(f)
        writer.writerow(table.field_names)# write the column name
        for record in table:# write the rows
            writer.writerow(list(record.values()))
    return csv_fn# return the csv name
```

Running the cell below will convert all dbf files in a given folder to csv format.

added "ignore_missing_memofile=True" to "table = DBF(infile, parserclass=MyFieldParser, ignore_missing_memofile=True)"

In [2]:
```
import fnmatch
import os
import csv
import time
import datetime
import sys
from dbfread import DBF, FieldParser, InvalidValue          # pip install d
bfread if needed

class MyFieldParser(FieldParser):
    def parse(self, field, data):
        try:
            return FieldParser.parse(self, field, data)
        except ValueError:
            return InvalidValue(data)


debugmode=0         # Set to 1 to catch all the errors.

for infile in os.listdir('.'):
    if fnmatch.fnmatch(infile, '*.dbf'):
        outfile = infile[:-4] + ".csv"
        print("Converting " + infile + " to " + outfile + ". Each period re
presents 2,000 records.")
        counter = 0
        starttime=time.clock()
        with open(outfile, 'w') as csvfile:
            table = DBF(infile, parserclass=MyFieldParser, ignore_missing_m
emofile=True)
            writer = csv.writer(csvfile)
            writer.writerow(table.field_names)
            for i, record in enumerate(table):
                for name, value in record.items():
                    if isinstance(value, InvalidValue):
                        if debugmode == 1:
                            print('records[{}][{!r}] == {!r}'.format(i, nam
e, value))
                writer.writerow(list(record.values()))
                counter +=1
                if counter%100000==0:
                    sys.stdout.write('!' + '\r\n')
                    endtime=time.clock()
#                    print (str("{:,}".format(counter))) + " records in "
 + str(endtime-starttime) + " seconds."
                elif counter%2000==0:
                    sys.stdout.write('.')
                else:
                    pass
        print("")
        endtime=time.clock()
        print ("Processed " + str("{:,}".format(counter)) + " records in "
+ str(endtime-starttime) + " seconds (" + str((endtime-starttime)/60) + " m
inutes.)")
        print (str(counter / (endtime-starttime)) + " records per second.")
        print("")
```

Running the function below will merge all csv files in the current folder into one pandas dataframe

In [4]:
```
import pandas as pd
import os

all_csv = [file_name for file_name in os.listdir(os.getcwd()) if '.csv' in
file_name]

li = []

for filename in all_csv:
    df = pd.read_csv(filename, index_col=None, header=0, parse_dates=True,
infer_datetime_format=True)
    li.append(df)

df = pd.concat(li, axis=0, ignore_index=True)
```