from collections import namedtuple

from collections import defaultdict

fields = namedtuple('fields', ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model\_year', 'origin', 'car\_name'])

import csv

import os

import requests

import logging

import argparse

from operator import attrgetter

import sys

import matplotlib.pyplot as plt

logger=logging.getLogger()

logger.setLevel(logging.DEBUG)

fileHandle = logging.FileHandler('autompg2.log', 'w')

fileHandle.setLevel(logging.DEBUG)

logger.addHandler(fileHandle)

streamHandle= logging.StreamHandler()

streamHandle.setLevel(logging.INFO)

logger.addHandler(streamHandle)

class AutoMPG:

"""Class that represents the attrributes available for each record in a data set"""

def \_\_init\_\_(self, make, model, year, mpg):

self.make=str(make)

self.model=str(model)

self.year=1900+int(year)

self.mpg=float(mpg)

def \_\_str\_\_(self):

#string representation of the record

return f'AutoMPG({self.make},{self.model}, {self.year}, {self.mpg})'

def \_\_repr\_\_(self):

#string representation of the record

return self.\_\_str\_\_()

def \_\_eq\_\_(self, other):

#compare the make model year and mpg of two records (equal)

#ensure types are the same

if type(self) == type(other):

return self.make == other.make and self.model == other.model and self.year==other.year and self.mpg==other.mpg

else:

return NotImplemented

def \_\_lt\_\_(self, other):

#compare the make model year and mpg of two records (less than)

#ensure types are the same

if type(self) == type(other):

#compare the first attribute: make

if self.make != other.make:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the second attribute: model

if self.model != other.model:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the third attribute: year

if self.year != other.year:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the fourth attribute: mpg

if self.mpg != other.mpg:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

#if all 4 attributes are the same, comparision not implemented

else:

return NotImplemented

else:

return NotImplemented

def \_\_hash\_\_(self):

#returns the hash of all the attributes

return hash((self.make, self.model, self.year, self.mpg))

class AutoMPGData:

"""Class that represents the entire AutoMPG data set. Data is a list of AutoMPG objects"""

def \_\_init\_\_(self):

self.data=self.\_load\_data()

def \_\_iter\_\_(self):

#allow iteration through the list

self.\_iter = 0

return self

def \_\_next\_\_(self):

#next is each record in the list

if self.\_iter == len(self.data):

raise StopIteration

ret = self.data[self.\_iter]

self.\_iter +=1

return ret

def \_load\_data(self):

#check if orginal data file exists

if os.path.exists("auto-mpg.data.txt")== False:

logging.debug('checking auto-mpg.data.txt')

self.\_get\_data()

#check if the file already exists

if os.path.exists("auto-mpg.clean.txt")== False:

logging.debug('checking auto-mpg.clean.txt')

#if file DNE, send to clean data

self.\_clean\_data()

else:

logging.debug('auto-mpg.clean.txt exits')

#used cleaned data to create the data list

data=[]

#take the data from the cleaned CSV

with open('auto-mpg.clean.txt', 'r', newline='') as csvfile:

dataReader = csv.reader(csvfile)

logging.debug('Parsing auto-mpg.clean.txt into AutoMPG objects')

for row in dataReader:

#put record into a tuple and unpack the tuple

field = fields(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8])

mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name = field

#add the record to the data list

car=Record(mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name)

data+=[car.pass\_into\_autoMPG()]

return data

def \_clean\_data(self):

#create a new data file with the cleaned data

#remove the tab

data\_file = csv.register\_dialect("space",delimiter='\t', skipinitialspace = True)

with open('auto-mpg.clean.txt', 'w', newline='') as outputfile:

cleanwriter = csv.writer(outputfile, quoting=csv.QUOTE\_NONNUMERIC)

#take the data from the original AutoMPG CSV file

with open('auto-mpg.data.txt', 'r', newline='') as csvfile:

dataReader = csv.reader(csvfile, "space")

for row in dataReader:

#split all attributes other than car\_name

clean=row[0].split() + [row[1]]

cleanwriter.writerow(clean)

self.\_load\_data()

def \_get\_data(self):

#access the data from website

logging.debug('getting auto-mpg.data.txt')

url='https://autompgdata.azurewebsites.net/api/assignment7data'

r = requests.get(url, allow\_redirects = True)

code=r.status\_code

logging.debug('Responce code: %s', r.status\_code)

#write the data from the website as a txt file

with open('auto-mpg.data.txt', 'wb') as output:

output.write(r.content)

self.\_load\_data()

return code

def sort\_by\_default(self):

#sort auto data by make model year then mpg

logging.debug('Sorting AutoMPG objects by default')

self.data.sort()

return self.data

def sort\_by\_year(self):

#sort auto data by year make model then mpg

logging.debug('Sorting AutoMPG objects by year')

self.data.sort(key= attrgetter('year', 'make', 'model', 'mpg'))

return self.data

def sort\_by\_mpg(self):

#sort auto data by mpg make model then year

logging.debug('Sorting AutoMPG objects by mpg')

self.data.sort(key= attrgetter('mpg', 'make', 'model', 'year'))

return self.data

def mpg\_by\_year(self):

#give the average mpg by car year

#define dictionaries

yearCounts=dict()

mpgSum= defaultdict(int)

mpgByYear= defaultdict(int)

f = attrgetter('make', 'model', 'year', 'mpg')

years=[]

for car in self.data:

#for each car count the cars of each year

year=f(car)[-2]

years+=[year]

#sum the mpg for each car per year

mpg=f(car)[-1]

mpgSum[year] += mpg

yearCounts = {year:years.count(year) for year in years}

#for each year divide the sum mpg by the count of cars that year

for year1, count in yearCounts.items():

for year2, MPGsum in mpgSum.items():

if year1==year2:

mpgByYear[year1] = (MPGsum/count)

return mpgByYear

def mpg\_by\_make(self):

#give the average mpg by car make

#define dictionaries

makeCounts=dict()

mpgSum= defaultdict(int)

mpgByMake= defaultdict(int)

f = attrgetter('make', 'model', 'year', 'mpg')

Makes=[]

for car in self.data:

#for each car count the cars of each make

make=f(car)[0]

Makes+=[make]

#sum the mpg for each car per make

mpg=f(car)[-1]

mpgSum[make] += mpg

makeCounts = {make:Makes.count(make) for make in Makes}

#for each make divide the sum mpg by the count of cars of that make

for make1, count in makeCounts.items():

for make2, MPGsum in mpgSum.items():

if make1==make2:

mpgByMake[make1] = (MPGsum/count)

return mpgByMake

class Record:

"""Class that passes the appropriate attributes into the AutoMPG class"""

def \_\_init\_\_(self, mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name):

self.mpg=mpg

self.cylinders=cylinders

self.displacement=displacement

self.horsepower=horsepower

self.weight=weight

self.acceleration=acceleration

self.model\_year=model\_year

self.origin=origin

self.car\_name=car\_name

def pass\_into\_autoMPG(self):

#split car\_name into make and model

split\_names=self.car\_name.split(" ", 1)

make=split\_names[0]

#if there is no model, add a blank as the model

if split\_names[-1]== split\_names[0]:

model=''

else:

model=split\_names[-1]

#correct spelling errors in data

make\_dict={"chevroelt":"chevrolet", "chevy":"chevrolet", "maxda":"mazda", "mercedes-benz":"mercedes", "toyouta": "toyota", "vokswagen":"volkswagen", "vw":"volkswagen"}

for key, value in make\_dict.items():

if make==key:

make=value

#pass attributes into AutoMPG

auto=AutoMPG(make,model,self.model\_year , self.mpg)

return auto

def main():

#initiate parser arguments

parser =argparse.ArgumentParser(description='Parse the data from specificed access log')

parser.add\_argument ('command', metavar='command' , type=str, help='command to execute', choices=['print', 'mpg\_by\_year', 'mpg\_by\_make'])

parser.add\_argument('-s', '--sort',metavar= '<sort order>', type=str, choices=[None, 'mpg', 'year'], help="sort the data")

parser.add\_argument('-o', '--ofile', metavar='<outfile>', type=str, help='specify output file')

parser.add\_argument('-p', '--plot', dest='do\_plot', action='store\_true', help='plot the data')

arguments= parser.parse\_args()

#call the AutoMPGdata class

car=AutoMPGData()

#sort if the command is print

if arguments.command=='print':

#default

if arguments.sort== None:

output=car.sort\_by\_default()

#mpg

if arguments.sort == 'mpg':

output=car.sort\_by\_mpg()

#year

if arguments.sort == 'year':

output=car.sort\_by\_year()

if arguments.ofile==None:

#if no file given print to stdout

list=[]

filewriter = csv.writer(sys.stdout, quoting=csv.QUOTE\_ALL)

filewriter.writerow(['make', 'model', 'year', 'mpg'])

#get each variable from the autompg object

f = attrgetter('make', 'model', 'year', 'mpg')

for car in output:

make=f(car)[0]

model=f(car)[1]

year=f(car)[2]

mpg=f(car)[-1]

list+=[make]+[model]+[year]+[mpg]

#print each car on a seperate line

for i in range(0,len(list),4):

filewriter.writerow(list[i:i+4])

if arguments.ofile!=None:

#if a file name was given

list=[]

with open(arguments.ofile, 'w', newline='') as outputfile:

filewriter = csv.writer(outputfile, quoting=csv.QUOTE\_ALL)

filewriter.writerow(['make', 'model', 'year', 'mpg'])

#get each variable from the autompg object

f = attrgetter('make', 'model', 'year', 'mpg')

for car in output:

make=f(car)[0]

model=f(car)[1]

year=f(car)[2]

mpg=f(car)[-1]

list+=[make]+[model]+[year]+[mpg]

#print each car on a seperate line

for i in range(0,len(list),4):

filewriter.writerow(list[i:i+4])

#if command is mpg by year

if arguments.command=='mpg\_by\_year':

output=car.mpg\_by\_year()

#if command is mpg by make

if arguments.command=='mpg\_by\_make':

output=car.mpg\_by\_make()

#if plot is called

if arguments.do\_plot:

keys=[]

values=[]

#take values out of dictionary, sort and put into list

dict\_tuples=sorted(output.items())

for i in dict\_tuples:

key=i[0]

value=i[1]

keys+=[key]

values+=[value]

#plot the key and values

plt.plot(keys,values, '--r')

plt.xticks(rotation = 75)

plt.show()

#write the output to stdout

if arguments.ofile==None:

filewriter = csv.writer(sys.stdout, quoting=csv.QUOTE\_ALL)

#take values out of dictionary, sort and put into list

dict\_tuples=sorted(output.items())

#write a line of the key and value

for i in dict\_tuples:

key=i[0]

value=i[1]

filewriter.writerow([key, value])

#write the output to specified file

if arguments.ofile!=None:

#take values out of dictionary, sort and put into list

dict\_tuples=sorted(output.items())

with open(arguments.ofile, 'w', newline='') as outputfile:

filewriter = csv.writer(outputfile, quoting=csv.QUOTE\_ALL)

#write a line of the key and value

for i in dict\_tuples:

key=i[0]

value=i[1]

filewriter.writerow([key, value])

if \_\_name\_\_=='\_\_main\_\_':

main()