# verifyBatteryChioce

from batteryModel import battery, logger  
import pandas as pd  
import numpy as np  
import threading  
import csv  
import json  
import time  
  
class calculate:  
 def \_\_init\_\_(self, avg\_data=True):  
 *# global config* self.log = logger("batteryChoiceEvaluation", 1000)self.max\_thread = 8  
 self.threadLimiter = threading.BoundedSemaphore(self.max\_thread)  
  
 self.base = "./"  
 iter\_file\_path = f"{self.base}iter.csv"  
 self.npy = f"{self.base}npy/"  
 self.plot = f"{self.base}plot/"  
  
 *# load consume and produce data* self.solarrad = np.load(self.npy + "solarrad.npy")  
 spring24 = np.load(self.npy + "spring24.npy")  
 summar24 = np.load(self.npy + "summer24.npy")  
 autumn24 = np.load(self.npy + "autumn24.npy")  
 winter24 = np.load(self.npy + "winter24.npy")  
 if avg\_data:  
 spring24 = np.sum(spring24, axis=0) / 1000  
 summar24 = np.sum(summar24, axis=0) / 1000  
 autumn24 = np.sum(autumn24, axis=0) / 1000  
 winter24 = np.sum(winter24, axis=0) / 1000  
 consume = []  
 for name in [spring24, summar24, autumn24, winter24]:  
 for i in range(80):  
 consume.append(list(name))  
 self.consume = np.array(consume).reshape(-1, ) / 1000  
  
 *# load iter data* self.iter\_data = pd.read\_csv(iter\_file\_path)  
  
 self.write\_list = []  
 with open("battery\_evaluate\_ouput.csv", "w") as f:  
 self.writter = csv.writer(f)  
 self.writter.writerow(  
 ["max\_nopower\_count", "Evaluate Score", "name", "max\_cap", "other\_info", "year\_round\_status",  
 "current\_cap"])  
  
 def evaluate(self, batteries=None, ret = False):  
 self.threadLimiter.acquire()  
 max\_nopower\_count = 0  
  
 if batteries is None:  
 batteries = [battery("default", 7000, 10000, 0.96, 100)]  
 *# self.log.write\_error("No battery option input try default instead")* current\_cap = []  
 status = True  
 year\_round\_status = []  
 for hour in range(320 \* 24):  
  
 *# extraxt data for power consume list and power produce list* charge = self.solarrad[hour]  
 discharge = self.consume[hour]  
 *# calculate current capacity of batteries & add list* sum\_cap = sum([i.capacity for i in batteries])  
 assert sum\_cap >= 0  
 current\_cap.append(sum\_cap)  
 try:  
 if charge > discharge:  
 pass  
 except Exception as e:  
 raise KeyboardInterrupt(f"\n{charge}\n{discharge}\n{e}")  
  
 if charge > discharge:  
 *# put power to battery if chaege larger than discharge* remain = charge - discharge  
 for bat in batteries: remain = bat.charge(remain)  
 *# reset status to true* status = True  
  
 elif charge < discharge:  
 *# pull power form battery if discharge larger than charge* pull = discharge - charge  
 for bat in batteries: pull = bat.discharge(pull)  
  
 *# if power can not suppply the use of user* if pull > 0:  
 *# self.log.write(f"{charge}, {discharge}, {sum([i.capacity for i in batteries])}, note: battery is later for one")  
 # self.log.write(f"No avaliable energy on day:{hour // 24} hour:{hour}")  
 # if not status: pass* if status:  
 max\_nopower\_count += 1  
 *# set status to false and max\_count -1* status = False  
 *# self.log.write( f"No avaliable energy on day:{hour // 24}, {sum([i.capacity for i in batteries])}, count\_increase, set status to false")* else:  
 self.log.write("balance")  
 if status:  
 year\_round\_status.append(1)  
 else:  
 year\_round\_status.append(0) *# "max\_nopower\_count", "Evaluate Score", "name","max\_cap","other\_info", "year\_round\_status", "current\_cap"* if max\_nopower\_count <= 10 and not ret:  
 score = self.choice\_analysis()  
 self.write\_list.append([max\_nopower\_count, score, json.dumps([i.name for i in batteries]),  
 sum([i.max\_capactity for i in batteries]), json.dumps([i.info() for i in batteries]),  
 json.dumps(year\_round\_status), json.dumps(current\_cap)])  
 self.threadLimiter.release()print("\t"+str(max\_nopower\_count), end=" ")  
 if ret: return year\_round\_status, current\_cap, max\_nopower\_count  
  
 def choice\_analysis(self):return 0  
  
 def checkCSVwritter(self):  
 *"""  
 todo: 检查文件导入顺序，是否与默认排序相同  
 :return:  
 """* while True:  
 time.sleep(1)  
 if len(self.write\_list) > 10:  
 with open("battery\_evaluate\_ouput.csv", "a+") as f:  
 write\_list\_copy = self.write\_list  
 self.write\_list = []  
 writer = csv.writer(f)  
 writer.writerows(write\_list\_copy)  
 def write\_all(self):  
 for i in range(self.max\_thread):  
 self.threadLimiter.acquire()  
 with open("battery\_evaluate\_ouput.csv", "a+") as f:  
 write\_list\_copy = self.write\_list  
 self.write\_list = []  
 writer = csv.writer(f)  
 writer.writerows(write\_list\_copy)  
  
 def main(self):name\_arrange = pd.read\_csv("iter.csv").values  
 battery\_data\_namedic = {}  
 for i in pd.read\_csv("BatteriesCSV2.csv").values:  
 battery\_data\_namedic[i[0]] = i[1:]  
 threading.Thread(target=self.checkCSVwritter).start()  
 threads = []  
 for f in range(len(name\_arrange)):  
 bats = []  
 for i in name\_arrange[f, 0].split(";")[:-1]:  
 obj = battery\_data\_namedic[i]  
 capacity = obj[5]  
 cost = obj[6]  
 efficiency = obj[4]  
 power = obj[7]  
 bats.append(battery(i, capacity, power, efficiency, cost))  
 self.threadLimiter.acquire()  
 t = threading.Thread(target=self.evaluate, args=(bats,))  
 threads.append(t)  
 print(f"\rStart thread {t}", end="")  
 t.start()  
 self.write\_all()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 c = calculate()  
 *# max\_recharge\_rate, max\_capactity, efficiency, cost=0, log=False* batteries = [battery(100, 100, 1)]  
 ret = c.evaluate(batteries=batteries, ret=True)  
 print(ret)

# batteryModel

import time, sys, threading  
  
  
class battery():  
 def \_\_init\_\_(self, max\_recharge\_rate, max\_capactity, efficiency, cost=0, log=False):  
 *"""  
 battery config and recharge rate  
 :param max\_recharge\_rate: float max recharge rate in w  
 :param max\_capactity: float unit in kw\*h/1000, max capactity  
 """* self.max\_recharge\_rate = max\_recharge\_rate  
 self.max\_capactity = max\_capactity  
 self.efficiency = efficiency  
 self.capacity = max\_capactity  
 self.log = All\_battery\_log  
 self.logstatus = log  
 self.log\_main = True  
 self.write\_to\_log = False  
 self.cost = cost  
 if log: print(  
 f"Battery set complied: max\_capactity = {self.capacity}\nmax\_recharge\_rate:{self.max\_recharge\_rate}, eff{efficiency}, cost{cost}")  
 self.log.write(f"Battery set complied: max\_capactity = {self.capacity}, max\_recharge\_rate:{self.max\_recharge\_rate}, efficiency:{efficiency},cost:{cost}")  
  
 def discharge(self, discharge\_target, log=False):if self.logstatus: log = True  
 if self.log\_main: self.log.write(f"Discharge {discharge\_target}， current cap : {self.capacity}")  
 discharge\_target\_real = discharge\_target / self.efficiency  
 if discharge\_target\_real <= self.max\_recharge\_rate:  
 if discharge\_target\_real <= self.capacity:  
 self.capacity -= discharge\_target\_real  
 return 0  
 else:  
 self.capacity = 0  
 if self.write\_to\_log :self.log.write(f"self.max\_recharge\_rate < discharge\_target\_real <= self.capacity: \t remain: {discharge\_target - (discharge\_target\_real - self.capacity) \* self.efficiency}",verbose=log)  
 return discharge\_target - self.capacity \* self.efficiency  
  
 else:  
 if self.max\_recharge\_rate <= self.capacity:  
 self.capacity -= self.max\_recharge\_rate  
 if self.write\_to\_log :self.log.write(f"discharge\_target\_real <= self.max\_recharge\_rate <= self.capacity: \t remain: {discharge\_target - self.max\_recharge\_rate \* self.efficiency}",verbose=log)  
 return discharge\_target - self.max\_recharge\_rate \* self.efficiency  
 else:  
 *# capacity\_current = self.capacity* self.capacity = 0  
 if self.write\_to\_log :self.log.write(f"disorder all cap\t remain:{discharge\_target - self.capacity \* self.efficiency}",verbose=log)  
 return discharge\_target - self.capacity \* self.efficiency  
  
 def charge(self, charge\_target, log=False):  
 if self.log\_main: self.log.write(f"Charge {charge\_target}， current cap : {self.capacity}")  
 charge\_target\_real = charge\_target \* self.efficiency  
 if self.logstatus: log = True  
 if charge\_target\_real >= self.max\_recharge\_rate:  
 if self.max\_recharge\_rate + self.capacity <= self.max\_capactity:  
 self.capacity += self.max\_recharge\_rate  
 if self.write\_to\_log: self.log.write(f"max\_rate < target & batter could not full \t remain:{charge\_target - self.max\_recharge\_rate / self.efficiency}",verbose=log)  
 return charge\_target - self.max\_recharge\_rate / self.efficiency  
 else:  
 capacity\_current = self.capacity  
 self.capacity = self.max\_capactity  
 if self.write\_to\_log: self.log.write(f"max\_rate < target & batter could full\t remain:{charge\_target - (self.max\_capactity - capacity\_current) / self.efficiency}",verbose=log)  
 return charge\_target - (self.max\_capactity - capacity\_current) / self.efficiency  
 else:  
 if charge\_target\_real + self.capacity <= self.max\_capactity:  
 self.capacity += charge\_target\_real  
 if self.write\_to\_log: self.log.write(f"max\_rate > target & could not full\t remain:{0}")  
 return 0  
 else:  
 current\_capactity = self.capacity  
 self.capacity = self.max\_capactity  
 if self.write\_to\_log: self.log.write(f"max\_rate > target & could full\t remain:{charge\_target - (self.max\_capactity - current\_capactity) / self.efficiency}",verbose=log)  
 return charge\_target - (self.max\_capactity - current\_capactity) / self.efficiency  
  
  
class logger():  
 def \_\_init\_\_(self, name = sys.argv[0], array\_limit = 10000):  
 self.logaddr = f"{name}.log"  
 self.verbose = True  
 self.array = []  
 self.array\_limit = array\_limit  
 with open(self.logaddr, "w") as f: f.write(  
 str(time.strftime("%H:%M:%S", time.localtime())) + str("\tLogger init\n"))  
  
 def write(self, content=None, verbose=False):  
 if verbose: print(content)  
 self.array.append(content)  
 if len(self.array) > self.array\_limit:  
 if verbose: print(f"update log at {self.logaddr}")  
 t = threading.Thread(target=self.append\_log)  
 t.start()  
  
 def append\_log(self):  
 array\_copy = self.array.copy()  
 self.array = []  
 with open(self.logaddr, "a+") as f:  
 for content in array\_copy:  
 f.write(str(str(time.strftime("%H:%M:%S", time.localtime())) + "\t[info]" + "\t" + content + "\n\n"))  
  
 def write\_error(self, error, content=None, verbose=False):  
 if verbose: print(error, content)  
 with open(self.logaddr, "a+") as f:  
 f.write(str(str(time.strftime("%H:%M:%S", time.localtime())) + "\t[error]" + str(error) + "\t" + str(  
 content) + "\n\n"))  
  
All\_battery\_log = logger("battery", 100000)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 libattery = battery(700, 10000, 0.96)  
 libattery.capacity = 200  
 for i in range(100):  
 libattery.charge(1000)  
 libattery.discharge(1000)

# batterChoiceEvaluation

*# author: Stevenli  
# time: 2021.11.12  
# final main V0.1*import csv  
import json  
import sys  
import threading  
import time  
  
import numpy as np  
import pandas as pd  
  
  
class calculate:  
 def \_\_init\_\_(self, avg\_data=True):  
 *# global config* self.log = logger("batteryChoiceEvaluation", 1000)  
 *# self.max\_recharge\_rate = 7000 # 最大充电速率7kw  
 # self.max\_solar\_generation\_rate = 20000 # 最大太阳能发20kwh* self.max\_thread = 8  
 self.threadLimiter = threading.BoundedSemaphore(self.max\_thread)  
  
 self.base = "./"  
 iter\_file\_path = f"{self.base}iter.csv"  
 self.npy = f"{self.base}npy/"  
 self.plot = f"{self.base}plot/"  
  
 *# load consume and produce data* self.solarrad = np.load(self.npy + "solarrad.npy")  
 spring24 = np.load(self.npy + "spring24.npy")  
 summar24 = np.load(self.npy + "summer24.npy")  
 autumn24 = np.load(self.npy + "autumn24.npy")  
 winter24 = np.load(self.npy + "winter24.npy")  
 if avg\_data:  
 spring24 = np.sum(spring24, axis=0) / 1000  
 summar24 = np.sum(summar24, axis=0) / 1000  
 autumn24 = np.sum(autumn24, axis=0) / 1000  
 winter24 = np.sum(winter24, axis=0) / 1000  
 consume = []  
 for name in [spring24, summar24, autumn24, winter24]:  
 for i in range(80):  
 consume.append(list(name))  
 self.consume = np.array(consume).reshape(-1, ) / 1000  
  
 *# load iter data* self.iter\_data = pd.read\_csv(iter\_file\_path)  
  
 self.write\_list = []  
 with open("battery\_evaluate\_ouput.csv", "w") as f:  
 self.writter = csv.writer(f)  
 self.writter.writerow(  
 ["max\_nopower\_count", "Evaluate Score", "name", "max\_cap", "other\_info", "year\_round\_status",  
 "current\_cap"])  
  
 def evaluate(self, batteries=None, ret = False):  
  
 max\_nopower\_count = 0  
  
 if batteries is None:  
 batteries = [battery("default", 7000, 10000, 0.96, 100)]  
 *# self.log.write\_error("No battery option input try default instead")* current\_cap = []  
 status = True  
 year\_round\_status = []  
 for hour in range(320 \* 24):  
  
 *# extraxt data for power consume list and power produce list* charge = self.solarrad[hour]  
 discharge = self.consume[hour]  
 *# calculate current capacity of batteries & add list* sum\_cap = sum([i.capacity for i in batteries])  
 assert sum\_cap >= 0  
 current\_cap.append(sum\_cap)  
 try:  
 if charge > discharge:  
 pass  
 except Exception as e:  
 raise KeyboardInterrupt(f"\n{charge}\n{discharge}\n{e}")  
  
 if charge > discharge:  
 *# put power to battery if chaege larger than discharge* remain = charge - discharge  
 for bat in batteries: remain = bat.charge(remain)  
 *# reset status to true* status = True  
  
 elif charge < discharge:  
 *# pull power form battery if discharge larger than charge* pull = discharge - charge  
 for bat in batteries: pull = bat.discharge(pull)  
  
 *# if power can not suppply the use of user* if pull > 0: *# if not status: pass* if status:  
 max\_nopower\_count += 1  
 *# set status to false and max\_count -1* status = False  
 *# self.log.write( f"No avaliable energy on day:{hour // 24}, {sum([i.capacity for i in batteries])}, count\_increase, set status to false")* else:  
 self.log.write("balance")  
 if status:  
 year\_round\_status.append(1)  
 else:  
 year\_round\_status.append(0)  
 *# raise KeyboardInterrupt("I am down")  
 # "max\_nopower\_count", "Evaluate Score", "name","max\_cap","other\_info", "year\_round\_status", "current\_cap"* if max\_nopower\_count <= 10:  
 score = self.choice\_analysis()  
 self.write\_list.append([max\_nopower\_count, score, json.dumps([i.name for i in batteries]),  
 sum([i.max\_capactity for i in batteries]), json.dumps([i.info() for i in batteries]),  
 json.dumps(year\_round\_status), json.dumps(current\_cap)])  
 self.threadLimiter.release()  
 *# print("End thread")* print("\t"+str(max\_nopower\_count), end=" ")  
 if ret: return year\_round\_status, current\_cap, max\_nopower\_count  
  
 def choice\_analysis(self):  
 *"""  
 todo：这里加大刘的程序  
 :return:   
 """* return 0  
  
 def checkCSVwritter(self):  
 *"""  
 todo: 检查文件导入顺序，是否与默认排序相同  
 :return:  
 """* while True:  
 time.sleep(1)  
 if len(self.write\_list) > 10:  
 with open("battery\_evaluate\_ouput.csv", "a+") as f:  
 write\_list\_copy = self.write\_list  
 self.write\_list = []  
 writer = csv.writer(f)  
 writer.writerows(write\_list\_copy)  
 def write\_all(self):  
 for i in range(self.max\_thread):  
 self.threadLimiter.acquire()  
 with open("battery\_evaluate\_ouput.csv", "a+") as f:  
 write\_list\_copy = self.write\_list  
 self.write\_list = []  
 writer = csv.writer(f)  
 writer.writerows(write\_list\_copy)  
  
 def main(self):  
 *"""  
 todo: 梳理结构，优化运行  
 todo：尝试使多线程返回东西  
 """* name\_arrange = pd.read\_csv("iter.csv").values  
 battery\_data\_namedic = {}  
 for i in pd.read\_csv("BatteriesCSV2.csv").values:  
 battery\_data\_namedic[i[0]] = i[1:]  
 threading.Thread(target=self.checkCSVwritter).start()  
 threads = []  
 for f in range(len(name\_arrange)):  
 bats = []  
 for i in name\_arrange[f, 0].split(";")[:-1]:  
 obj = battery\_data\_namedic[i]  
 capacity = obj[5]  
 cost = obj[6]  
 efficiency = obj[4]  
 power = obj[7]  
 bats.append(battery(i, capacity, power, efficiency, cost))  
 self.threadLimiter.acquire()  
 t = threading.Thread(target=self.evaluate, args=(bats,))  
 threads.append(t)  
 print(f"\rStart thread {t}", end="")  
 t.start()  
 self.write\_all()  
  
  
class battery:  
 def \_\_init\_\_(self, name, max\_recharge\_rate, max\_capactity, efficiency, cost, log=False):  
 *"""  
 todo: 优化导入和battery的注释  
 todo: 检查当前充放电逻辑  
 battery config and recharge rate  
 :param max\_recharge\_rate: float max recharge rate in w  
 :param max\_capactity: float unit in kw\*h/1000, max capactity  
 """* self.name = name  
 self.max\_recharge\_rate = max\_recharge\_rate  
 self.max\_capactity = max\_capactity  
 self.efficiency = efficiency  
 assert type(efficiency) == float  
 self.capacity = max\_capactity  
 self.log = All\_battery\_log  
 self.logstatus = log  
 self.log\_main = False  
 self.write\_to\_log = False  
 self.cost = cost  
 if log: print(  
 f"Battery set complied: max\_capactity = {self.capacity}\nmax\_recharge\_rate:{self.max\_recharge\_rate}, eff{efficiency}, cost{cost}")  
 self.log.write(  
 f"Battery set complied: max\_capactity = {self.capacity}, max\_recharge\_rate:{self.max\_recharge\_rate}, efficiency:{efficiency},cost:{cost}")  
  
 def info(self):  
 return {  
 "name": self.name,  
 "max\_recharge\_rate": self.max\_recharge\_rate,  
 "max\_capactity": self.max\_capactity,  
 "efficiency": self.efficiency,  
 "cost": self.cost,  
 }  
  
 def discharge(self, discharge\_target, log=False):  
 *# discharge\_target\_real: 真实的电池放电，但是target减少self.efficiency倍  
 # discharge\_target: 用电器计划从电池要走的电量* if self.logstatus: log = True  
 if self.log\_main: self.log.write(f"Discharge {discharge\_target}， current cap : {self.capacity}")  
  
 discharge\_target\_real = discharge\_target / self.efficiency  
  
 if discharge\_target\_real <= self.max\_recharge\_rate:  
 if discharge\_target\_real <= self.capacity:  
 self.capacity -= discharge\_target\_real  
 return 0  
 else:  
 self.capacity = 0  
 if self.write\_to\_log: self.log.write(  
 f"self.max\_recharge\_rate < discharge\_target\_real <= self.capacity: \t remain: {discharge\_target - (discharge\_target\_real - self.capacity) \* self.efficiency}",  
 verbose=log)  
 return discharge\_target - self.capacity \* self.efficiency  
  
 else:  
 if self.max\_recharge\_rate <= self.capacity:  
 self.capacity -= self.max\_recharge\_rate  
 if self.write\_to\_log: self.log.write(  
 f"discharge\_target\_real <= self.max\_recharge\_rate <= self.capacity: \t remain: {discharge\_target - self.max\_recharge\_rate \* self.efficiency}",  
 verbose=log)  
 return discharge\_target - self.max\_recharge\_rate \* self.efficiency  
 else:  
 *# capacity\_current = self.capacity* self.capacity = 0  
 if self.write\_to\_log: self.log.write(  
 f"disorder all cap\t remain:{discharge\_target - self.capacity \* self.efficiency}", verbose=log)  
 return discharge\_target - self.capacity \* self.efficiency  
  
 def charge(self, charge\_target, log=False):  
 if self.log\_main: self.log.write(f"Charge {charge\_target}， current cap : {self.capacity}")  
 charge\_target\_real = charge\_target \* self.efficiency  
 if self.logstatus: log = True  
 if charge\_target\_real >= self.max\_recharge\_rate:  
 if self.max\_recharge\_rate + self.capacity <= self.max\_capactity:  
 self.capacity += self.max\_recharge\_rate  
 if self.write\_to\_log: self.log.write(  
 f"max\_rate < target & batter could not full \t remain:{charge\_target - self.max\_recharge\_rate / self.efficiency}",  
 verbose=log)  
 return charge\_target - self.max\_recharge\_rate / self.efficiency  
 else:  
 capacity\_current = self.capacity  
 self.capacity = self.max\_capactity  
 if self.write\_to\_log: self.log.write(  
 f"max\_rate < target & batter could full\t remain:{charge\_target - (self.max\_capactity - capacity\_current) / self.efficiency}",  
 verbose=log)  
 return charge\_target - (self.max\_capactity - capacity\_current) / self.efficiency  
 else:  
 if charge\_target\_real + self.capacity <= self.max\_capactity:  
 self.capacity += charge\_target\_real  
 if self.write\_to\_log: self.log.write(f"max\_rate > target & could not full\t remain:{0}")  
 return 0  
 else:  
 current\_capactity = self.capacity  
 self.capacity = self.max\_capactity  
 if self.write\_to\_log: self.log.write(  
 f"max\_rate > target & could full\t remain:{charge\_target - (self.max\_capactity - current\_capactity) / self.efficiency}",  
 verbose=log)  
 return charge\_target - (self.max\_capactity - current\_capactity) / self.efficiency  
  
  
class logger:  
 def \_\_init\_\_(self, name=sys.argv[0], array\_limit=10000):  
 self.logaddr = f"{name}.log"  
 self.verbose = True  
 self.array = []  
 self.array\_limit = array\_limit  
 with open(self.logaddr, "w") as f: f.write(  
 str(time.strftime("%H:%M:%S", time.localtime())) + str("\tLogger init\n"))  
  
 def write(self, content=None, verbose=False):  
 if verbose: print(content)  
 self.array.append(content)  
 if len(self.array) > self.array\_limit:  
 if verbose: print(f"update log at {self.logaddr}")  
 t = threading.Thread(target=self.append\_log)  
 t.start()  
  
 def append\_log(self):  
 array\_copy = self.array.copy()  
 self.array = []  
 with open(self.logaddr, "a+") as f:  
 for content in array\_copy:  
 f.write(str(str(time.strftime("%H:%M:%S", time.localtime())) + "\t[info]" + "\t" + content + "\n\n"))  
  
 def write\_error(self, error, content=None, verbose=False):  
 if verbose: print(error, content)  
 with open(self.logaddr, "a+") as f:  
 f.write(str(str(time.strftime("%H:%M:%S", time.localtime())) + "\t[error]" + str(error) + "\t" + str(  
 content) + "\n\n"))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 """  
 :todo: 梳理日志写法，梳理文件结构  
 """  
 All\_battery\_log = logger("battery", 100000)  
 calculate().main()

# solarIteration

*#!/usr/bin/env python  
# coding: utf-8  
  
# In[38]:*import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
get\_ipython().run\_line\_magic('matplotlib', 'inline')  
import itertools as it  
import csvbatteryConfig = pd.read\_csv("BatteriesCSV.csv")  
batteryConfigiteration = []  
with open("iter.csv", "w") as f:  
 writer = csv.writer(f)  
 writer.writerow(batteryConfig.keys())  
 for i in range(1,len(batteryConfig)+1):  
 for e in it.combinations\_with\_replacement(batteryConfig.values, i):  
 if len(e) == 1:  
 writer.writerows(e)  
 else:  
 writer.writerow(np.sum(e, axis=0))

# solarDataAnalysis

*#!/usr/bin/env python  
# coding: utf-8*import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import scipy.interpolate  
import random  
  
plot = "plot/"  
npy = "npy/"solar\_rad = pd.read\_csv("xunlian.csv")  
solar\_radsolar\_rad\_array = np.array(solar\_rad)  
solar\_rad\_value\_array = np.array(solar\_rad\_array[:,2], dtype=int)/max(solar\_rad\_array[:,2])\*20000max\_per\_hour = []  
for value in range(len(solar\_rad\_value\_array)//60):  
 max\_per\_hour.append(list(solar\_rad\_value\_array[60\*value:60\*value+60]))  
data = np.max(max\_per\_hour, axis=1)  
color = np.array(data, dtype=int)  
plt.scatter(np.arange(len(data)), data, c=np.array(data), s=0.4)  
plt.title("MaxPerHourPlot")  
plt.savefig(f"{plot}MaxPerHourPlot.png",format='jpg', dpi=2400)  
plt.show()  
plt.scatter(np.arange(len(data)), np.sort(data), c=np.sort(np.array(data)), s=0.1)  
plt.title(f"{plot}SortedOfMaxPerHourPlot")  
plt.savefig(f"{plot}SortedOfMaxPerHourPlot.png",format='jpg', dpi=2400)  
plt.show()max\_per\_day = []  
for value in range(len(solar\_rad\_value\_array)//(60\*24)):  
 max\_per\_day.append(list(solar\_rad\_value\_array[(60\*24)\*value:(60\*24)\*value+(60\*24)]))  
data = np.max(max\_per\_day, axis=1)  
color = np.array(data, dtype=int)  
plt.scatter(np.arange(len(data)), data, c=np.array(data), s=4)  
plt.title("MaxPerDayPlot")  
plt.savefig(f"{plot}MaxPerDayPlot.png")  
plt.show()  
plt.scatter(np.arange(len(data)), np.sort(data), c=np.sort(np.array(data)), s=2)  
plt.title("SortedOfMaxPerDayPlot")  
plt.savefig(f"{plot}SortedOfMaxPerDayPlot.png")  
plt.show()max\_per\_mouth = []  
for value in range(len(solar\_rad\_value\_array)//(60\*24\*30)):  
 max\_per\_mouth.append(list(solar\_rad\_value\_array[(60\*24\*30)\*value:(60\*24\*30)\*value+(60\*24\*30)]))  
data = np.max(max\_per\_mouth, axis=1)  
color = np.array(data, dtype=int)  
plt.plot(data)  
plt.title("MaxPermouthPlot")  
plt.savefig(f"{plot}MaxPermouthPlot.png",format='jpg', dpi=2400)  
plt.show()  
plt.plot(np.sort(data))  
plt.title("SortedOfMaxPermouthPlot")  
plt.savefig(f"{plot}SortedOfMaxPermouthPlot.png",format='jpg', dpi=2400)  
plt.show()  
x\_new = np.linspace(0, len(data), 300)  
a\_BSpline = scipy.interpolate.make\_interp\_spline(np.arange(len(data)), data)  
y\_new = a\_BSpline(x\_new)  
plt.plot(x\_new, y\_new)  
plt.savefig(f"{plot}SmoothedMaxPermouthPlot.png",format='jpg', dpi=2400)  
plt.show()color = np.array(solar\_rad.values[:,2]/max(solar\_rad.values[:,2])\*1000, dtype=int)  
plt.scatter(np.arange(len(solar\_rad)), np.sort(np.array(solar\_rad.values[:,2]), -1), c=np.sort(color, -1), s=0.1)  
plt.title("Solar radiation ranking table")  
plt.savefig(f"{plot}SolarRadiationRankingTable.png",format='jpg', dpi=2400)  
plt.show()  
  
  
*# In[ ]:*max\_per\_halfMouth = []  
for value in range(len(solar\_rad\_value\_array)//(60\*24\*10)):  
 max\_per\_halfMouth.append(list(solar\_rad\_value\_array[(60\*24\*10)\*value:(60\*24\*10)\*value+(60\*24\*10)]))  
data = np.max(np.array(max\_per\_halfMouth), axis=1)  
color = np.array(data, dtype=int)  
plt.plot(data)  
plt.title("MaxPerhalfMouthPlot")  
plt.savefig(f"{plot}MaxPerhalfMouthPlot.png",format='jpg', dpi=2400)  
plt.show()  
plt.plot(np.sort(data))  
plt.title("SortedOfMaxPerhalfMouthPlot")  
plt.savefig(f"{plot}SortedOfMaxPerhalfMouthPlot.png",format='jpg', dpi=2400)  
plt.show()  
x\_new = np.linspace(0, len(data), 300)  
a\_BSpline = scipy.interpolate.make\_interp\_spline(np.arange(len(data)), data)  
y\_new = a\_BSpline(x\_new)  
plt.plot(x\_new, y\_new)  
plt.savefig(f"{plot}SmoothedMaxPerhalfMouthPlot.png",format='jpg', dpi=2400)  
plt.show()sum\_per\_Mouth = []  
for value in range(len(solar\_rad\_value\_array)//(60\*24\*10)):  
 sum\_per\_Mouth.append(list(solar\_rad\_value\_array[(60\*24\*10)\*value:(60\*24\*10)\*value+(60\*24\*10)]))  
data = np.sum(np.array(sum\_per\_Mouth), axis=1)  
color = np.array(data, dtype=int)  
plt.plot(data)  
plt.title("SumPer10daysPlot")  
plt.savefig(f"{plot}SumPer10daysPlot.png",format='png', dpi=6000)  
plt.show()  
plt.plot(np.sort(data))  
plt.title("SortedOfSumPer10daysPlot")  
plt.savefig(f"{plot}SortedOfSumPer10daysPlot.png",format='png', dpi=6000)  
plt.show()  
x\_new = np.linspace(0, len(data), 300)  
a\_BSpline = scipy.interpolate.make\_interp\_spline(np.arange(len(data)), data)  
y\_new = a\_BSpline(x\_new)  
plt.plot(x\_new, y\_new)  
plt.savefig(f"{plot}SmoothedSumPer10daysPlot.png",format='png', dpi=6000)  
plt.show()  
  
  
*# In[ ]:*sum\_per\_mouth = []  
for value in range(len(solar\_rad\_value\_array)//(60\*24\*30)):  
 sum\_per\_mouth.append(list(solar\_rad\_value\_array[(60\*24\*30)\*value:(60\*24\*30)\*value+(60\*24\*30)]))  
data = np.sum(sum\_per\_mouth, axis=1)  
color = np.array(data, dtype=int)  
plt.plot(data)  
plt.title("sumPermouthPlot")  
plt.savefig(f"{plot}sumPermouthPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
plt.plot(np.sort(data))  
plt.title("SortedOfsumPermouthPlot")  
plt.savefig(f"{plot}SortedOfsumPermouthPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
x\_new = np.linspace(0, len(data), 300)  
a\_BSpline = scipy.interpolate.make\_interp\_spline(np.arange(len(data)), data)  
y\_new = a\_BSpline(x\_new)  
plt.plot(x\_new, y\_new)  
plt.title("SmoothedsumPermouthPlot")  
plt.savefig(f"{plot}SmoothedsumPermouthPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
  
  
*# In[ ]:*sum\_per\_day = []  
for value in range(len(solar\_rad\_value\_array)//(60)):  
 sum\_per\_day.append(list(solar\_rad\_value\_array[(60)\*value:(60)\*value+(60)]))  
data = np.sum(sum\_per\_day, axis=1)  
solar\_rad\_value\_array = data/max(data)\*20000  
  
  
*# In[ ]:*choice = random.randint(0, len(data)//24)\*24  
data = data/max(data)\*20000  
day\_data = data[choice:choice+24]  
  
plt.plot(day\_data)  
plt.title("sumOnedayPlot")  
plt.savefig(f"{plot}sumOnedayPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
plt.plot(np.sort(day\_data))  
plt.title("SortedOfsumOnedayPlot")  
plt.savefig(f"{plot}SortedOfsumOnedayPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
x\_new = np.linspace(0, len(day\_data), 300)  
a\_BSpline = scipy.interpolate.make\_interp\_spline(np.arange(len(data)), data)  
y\_new = a\_BSpline(x\_new)  
plt.plot(x\_new, y\_new)  
plt.title("SmoothedsumOnedayPlot")  
plt.savefig(f"{plot}SmoothedsumOnedayPlot.jpg",format='jpg', dpi=2400)  
plt.show()  
  
  
*# In[ ]:*# dailyElectricEstimate

*#!/usr/bin/env python  
# coding: utf-8  
  
# # 预估每天的电能消耗  
#   
# ## 通过估算每天的电能消耗，并加上了正态分布的噪音值来生成每天的估算，生成了四季每天的预估数值。  
#   
# author: 李树雨<br>time: 2021.11.6  
#   
  
# In[1]:*import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import random  
get\_ipython().run\_line\_magic('matplotlib', 'inline')  
  
  
*# In[2]:  
  
  
# global Settings  
#numpy normal 取值数量*numpy\_normal\_sample\_number = 1000  
numpy\_normal\_divided = 2  
  
*# 文件夹管理*base\_dir = "/Users/lishuyu/PycharmProjects/solarPowerEstimate/"  
plotDIR = "plot/"  
npyDIR = "npy/"  
get\_ipython().system('mkdir plot')  
get\_ipython().system('mkdir npy')  
powerConsumingData = pd.read\_csv("solar1.csv")  
status = powerConsumingData["spring"].values  
powerConsumingData  
 *# 台数*number = []  
for i in powerConsumingData["台数"]:  
 min\_val = int(i.split(",")[0])  
 max\_val = int(i.split(",")[1])  
 cache = np.random.normal((min\_val+max\_val)/2, (max\_val-min\_val)/2/numpy\_normal\_divided, numpy\_normal\_sample\_number)  
 number.append([i if i>0 else 0 for i in cache])  
number = np.array(number)  
*# 运行时长*time = []  
for i in powerConsumingData.values:  
 cache = np.random.normal(i[2], i[4]/numpy\_normal\_divided, numpy\_normal\_sample\_number)  
 time.append([i if i>0 else 0 for i in cache])  
time = np.array(time)  
*# 额定功率*watt = []  
for i in range(numpy\_normal\_sample\_number):  
 watt.append(powerConsumingData["额定功率"])  
watt = np.array(watt).TWattPerDay = number \* time \* watt  
powerLabel = ["Computer","washing machine","hairdryer","electric kettle" , "disinfection cabinet" , "water heater" ,"electric iron" , "TV" , "refrigerator" ,"electric lamp DAY", "electric lamp NIGHT" , "rice cooker" , "Air conditioner" , "microwave oven" , "Bath bar","electric stove"]  
plt.plot(WattPerDay, powerLabel)  
plt.xlabel("KW\*H")  
plt.ylabel("class")  
plt.grid()  
plt.savefig(f"{plotDIR}class\_KW\*h\_plot.png")  
plt.show()  
 *#Averge per day MAX*np.sum(WattPerDay)/1000/1000  
def extend\_npsample(season):  
 global powerConsumingData, numpy\_normal\_sample\_number  
 status = []  
 for i in range(numpy\_normal\_sample\_number):  
 status.append(powerConsumingData[season])  
 return np.array(status).T  
  
spring = extend\_npsample("spring") \* WattPerDay  
summer = extend\_npsample("summar") \* WattPerDay  
autumn = extend\_npsample("autum") \* WattPerDay  
winter = extend\_npsample("winter") \* WattPerDay  
  
SeanonData = np.array([np.sum(spring), np.sum(summer), np.sum(autumn), np.sum(winter)])/1000/numpy\_normal\_sample\_number  
plt.plot(SeanonData)  
plt.ylabel("kw\*h")  
plt.grid()  
plt.savefig(f"{plotDIR}seasonAverage.png")  
plt.show()  
  
  
*# In[13]:  
  
  
# #Averge per day real*(np.sum(spring) + np.sum(summer) + np.sum(autumn) + np.sum(winter))/1000/numpy\_normal\_sample\_number/4  
  
  
*# In[14]:*plt.boxplot([np.sum(spring, axis=0)/1000,np.sum(summer, axis=0)/1000,np.sum(autumn, axis=0)/1000,np.sum(winter, axis=0)/1000])  
plt.savefig(f"{plotDIR}boxplotEachseason.png")  
plt.show()  
  
  
*# In[15]:*def calCurrent24HourEnergyConsuming(seasonData):  
 startTime, endTime, duration = powerConsumingData.values[:,9], powerConsumingData.values[:,10], powerConsumingData.values[:,2]  
 power24ConsumeList = []  
 for seasonDataIterater in range(numpy\_normal\_sample\_number):  
 energyConsume = seasonData[:,seasonDataIterater]  
 power24Consuming = [0 for i in range(24)]  
 for i in range(len(startTime)):  
 *# print(f"Current on i = {i}")* time = random.randint(startTime[i],endTime[i])  
 for currentHour in range(time, time + int(duration[i] - 0.01) + 1, 1):  
 if currentHour >= 24: currentHour -= 24  
 power24Consuming[currentHour-1] += energyConsume[i]/duration[i]  
 *# print(f"Add {currentHour}")* power24ConsumeList.append(power24Consuming)  
 return np.array(power24ConsumeList)  
  
  
*# In[16]:*spring24 = calCurrent24HourEnergyConsuming(spring)  
summer24 = calCurrent24HourEnergyConsuming(summer)  
autumn24 = calCurrent24HourEnergyConsuming(autumn)  
winter24 = calCurrent24HourEnergyConsuming(winter)  
  
  
*# In[17]:*plt.boxplot(spring24, showfliers=False)  
plt.title("spring24")  
plt.savefig(f"{plotDIR}spring24.png")  
plt.show()  
plt.boxplot(summer24, showfliers=False)  
plt.title("summer24")  
plt.savefig(f"{plotDIR}summer24.png")  
plt.show()  
plt.boxplot(autumn24, showfliers=False)  
plt.title("autumn24")  
plt.savefig(f"{plotDIR}autumn24.png")  
plt.show()  
plt.boxplot(winter24, showfliers=False)  
plt.title("winter24")  
plt.savefig(f"{plotDIR}winter24.png")  
plt.show()  
  
  
*# In[18]:*np.save(f"{npyDIR}spring24.npy", spring24)  
np.save(f"{npyDIR}summer24.npy", summer24)  
np.save(f"{npyDIR}autumn24.npy", autumn24)  
np.save(f"{npyDIR}winter24.npy", winter24)  
  
  
*# In[ ]:*

# The batteryanalysis.py program arranges all possible battery combinations and adds up their data to produce a list. The batteryAnalysis2.py program filters through a list of battery combinations based on what the user prefers and the criteria specified by the user.

import pandas

data=pandas.read\_csv("/Users/dayouliu/Desktop/Batteries.csv")

batterynuber=float(input("battery number"))

dataValue = data.values

lista=[]

listb=[]

name=["data","batname","number"]

if batterynuber==2 :

for i in range(10):

c=float(dataValue[i,1])

for v in range(9-i):

d=float(dataValue[v+i,1])

Cost=(c+d)/2

print("avarage cost",dataValue[i,0],dataValue[v+i,0],Cost)

lista.append("cost")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(Cost)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,2])

for v in range(9-i):

d=float(dataValue[v+i,2])

f=(c+d)/2

print("avarage energy density",dataValue[i,0],dataValue[v+i,0],f)

lista.append("AED")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,3])

for v in range(9-i):

d=float(dataValue[v+i,3])

f=(c+d)/2

print("average effiecent",dataValue[i,0],dataValue[v+i,0],f)

lista.append("AE")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,4])

for v in range(9-i):

d=float(dataValue[v+i,4])

f=(c+d)/2

print("avarage KW",dataValue[i,0],dataValue[v+i,0],f)

lista.append("AKW")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,5])

for v in range(9-i):

d=float(dataValue[v+i,5])

f=(c+d)/2

print("avarage life",dataValue[i,0],dataValue[v+i,0],f)

lista.append("Al")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,6])

for v in range(9-i):

d=float(dataValue[v+i,6])

f=(c+d)

print("avarage life",dataValue[i,0],dataValue[v+i,0],f)

lista.append("Al")

lista.append(dataValue[i,0]+dataValue[v+i,0])

lista.append(f)

listb.append(lista)

lista=[]

if batterynuber==3:

for i in range(10):

c=float(dataValue[i,7])

for v in range(9-i):

d=float(dataValue[v+i,7])

for n in range(9-v):

l=float(dataValue[n+v,7])

f=c+d+l

lista.append("Ac")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,3])

for v in range(9-i):

d=float(dataValue[v+i,3])

for n in range(9-v):

l=float(dataValue[n+v,3])

f=(c+d+l)/3

lista.append("Aed")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,4])

for v in range(9-i):

d=float(dataValue[v+i,4])

for n in range(9-v):

l=float(dataValue[n+v,4])

f=(c+d+l)/3

lista.append("eff")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,5])

for v in range(9-i):

d=float(dataValue[v+i,5])

for n in range(9-v):

l=float(dataValue[n+v,5])

f=(c+d+l)/3

lista.append("kw")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,6])

for v in range(9-i):

d=float(dataValue[v+i,6])

for n in range(9-v):

l=float(dataValue[n+v,6])

f=(c+d+l)/3

lista.append("Al")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

for i in range(10):

c=float(dataValue[i,8])

for v in range(9-i):

d=float(dataValue[v+i,8])

for n in range(9-v):

l=float(dataValue[n+v,8])

f=(c+d+l)/3

lista.append("power")

lista.append(dataValue[i,0]+dataValue[v+i,0]+dataValue[n+v,0])

lista.append(f)

listb.append(lista)

lista=[]

test=pandas.DataFrame(columns=name,data=listb)

test.to\_csv("/Users/dayouliu/Desktop/sb2.csv")

**# another program**

import pandas

data=pandas.read\_csv("/Users/dayouliu/Desktop/sbsbsbsbsb.csv")

data2=pandas.read\_csv("/Users/dayouliu/Desktop/batteries3.csv")

cost=float(input("cost of batt"))

capacity=float(input("capacity"))

power=float(input("power"))

b=0

dataValue = data.values

dataValue2=data2.values

for i in range(len(dataValue)):

if dataValue[i,8] <= cost and dataValue[i,7]\*2 >= capacity and dataValue[i,9]>=power/2:

print("avilable batteries are: ",dataValue[i,2])

b=(dataValue[i,8]/661)\*-11 +(dataValue[i,4]/0.08427)\*0.1+(dataValue[i,5]/6.522)\*0.3+(dataValue[i,6]/0.9)\*0.2

b=b/0.9

print("best choices are: ",dataValue[i,2],b)

for i in range(len(dataValue2)):

if dataValue2[i,3] <= cost and dataValue2[i,7]\*3 >= capacity and dataValue2[i,8]>=power/3:

print("avilable batteries are: ",dataValue2[i,2])

c=(dataValue2[i,3]/661\*3)\*-11+(dataValue2[i,4]/0.08427)\*0.1+(dataValue2[i,5]/6.522)\*0.3+(dataValue2[i,6]/0.9)\*0.2

c=c/0.9

print("best choices are: ",dataValue2[i,2],c)