Guzman Homework

November 17, 2020

1. **Assignment One: Power Calendar function**
   The goal of this task is to generate the specific type of hours given ISO, peak type and period. I create a class for the whole process. The class is gethour, which contains several attributes including some necessary attributes like iso, peak type, period, start date, end date, hour and some supportive attributes like year, iseastern, holiday, isdaylight ,and duration.
   In order to calculation the peak days, I create a dictionary to store all NERC holidays. The calculation of some holidays is tricky. For example, New Year's Day is usually Jan 1st in a year. However, it may be Jan 2nd in some years when the Jan 1st lands on Sunday.I don't ensure about this property for New Year's Day. I look at the dates of New Year's Day for different years and then imply this property.The Christmas Day is confirmed by Wikipedia for this pattern.
   Identifying the daylight dates is also vital for calculating different peak types. The daylight date in March is the second Sunday in March. In this date, we need have only 23 hours. The daylight date in November is the first Sunday in November. In this date, we need have 25 hours.
   I divide the hours calculation into four parts based on eastern/western power station and whether the iso follow daylight, given eastern power station has different definition for weekdays. Even though MISO is the only power station that does not have the daylight-saving setting, I leave some space in the code for further modification.

```
import datetime
import calendar

class get_hours(object):
    """
    generate power calendar given the iso, peak type and period
    """

    def __init__(self, iso="ERCOT",peak_type="flat",period="2021A"):
        self.iso = iso
        self.peak_type=peak_type
        self.period=period
        self.year=0 # store the year of the period
        self.period_type=self.period_tp(period) # store period type like daily, monthly
    , quarterly and annually
        self.start_date=self.start_dat(period)
        self.end_date=self.end_dat(period)
        self.is_eastern=self.is_Eastern(iso) # check if the iso belongs to eastern
    power station
        self.holiday=self.NERC_holiday() # create a dictionary to store all NERC
    holiday for the year
        self.is_daylight=self.is_daylight_setting() #check if the iso follows the
    daylight setting
        self.duration=(self.end_date-self.start_date).days+1 # check how many days for
    the input
        self.hour=self.gethours()


    def period_tp(self, period):
        """
        get the period type
        """
        if period[-1]=="A":
```

```
            return "annually"
        elif "-" in period:
            return "daily"
        elif period[-2]=="Q":
            return "quarterly"
        else:
            return "monthly"

    def start_dat(self, period):
        """
        get the start date given the period
        """
        if self.period_type == "annually":
            self.year=int(period[:-1])
            return datetime.date(int(period[:-1]),1,1)
        elif self.period_type == "daily":
            year=int(period.split("-")[0])
            self.year=year
            month=int(period.split("-")[1])
            date=int(period.split("-")[2])
            return datetime.date(year,month,date)
        elif self.period_type == "quarterly":
            year=int(period.split("Q")[0])
            self.year=year
            quarter=int(period.split("Q")[1])
            return datetime.date(year,quarter*3-2,1)
        else:
            year=int(period[:-3])
            self.year=year
            month_abbr=period[-3:]
            month_num=list(calendar.month_abbr).index(month_abbr)
            return datetime.date(year,month_num,1)

    def end_dat(self, period):
        """
        get the end date given the period
        """
        if self.period_type == "annually":
            return datetime.date(int(period[:-1]),12,31)
        elif self.period_type == "daily":
            year=int(period.split("-")[0])
            month=int(period.split("-")[1])
            date=int(period.split("-")[2])
            return datetime.date(year,month,date)
        elif self.period_type == "quarterly":
            year=int(period.split("Q")[0])
            quarter=int(period.split("Q")[1])
            if quarter in [1,4]: # there are 31 days in the final month of quarter 1
    and 4
                return datetime.date(year,quarter*3,31)
            else:
                return datetime.date(year,quarter*3,30)
        else:
            year=int(period[:-3])
            month_abbr=period[-3:]
            month_num=list(calendar.month_abbr).index(month_abbr) #transfer the month
    string to month number
            _ ,date =calendar.monthrange(year,month_num)
            return datetime.date(year,month_num,date)

    def is_Eastern(self, iso):
        """
        check if the iso belongs to eastern power market
        """
        if iso in ["PJM","MISO","ERCOT","SPP","NYISO"]:
            return True
        elif iso in ["WECC","CAISO"]:
            return False
```

```
        else:
            raise NameError("Please make sure the input is one of the seven iso")

    def NERC_holiday(self):
        """
        create a dictionary to store all NERC holiday for the year
        """
        holiday_dict={}

        ## Memorial, last Monday of May
        cal = calendar.Calendar(firstweekday=0)
        month = cal.monthdatescalendar(self.year, 5)
        lastweek = month[-1]
        monday_mem = lastweek[0]
        holiday_dict["Memorial"]=monday_mem

        ## Independence
        holiday_dict["Independence"]=datetime.date(self.year,7,4)

        ## Labor, first Monday of Sep
        month = cal.monthdatescalendar(self.year, 9)
        lastweek = month[0]
        monday_la = lastweek[0]
        if monday_la.month !=9:
            monday_la=month[1][0]
        holiday_dict["Labor"]=monday_la

        ## Thanksgiving, fourth Thursday of Nov
        cal = calendar.Calendar(firstweekday=0)
        month = cal.monthdatescalendar(self.year, 11)
        firstweek = month[0]
        first_thrusday = firstweek[3]
        if first_thrusday.month !=11:
            fourth_tuesday=month[4][3]
        else:
            fourth_tuesday=month[3][3]
        holiday_dict["Thanksgiving"]=fourth_tuesday

        ## New year, the first date of the year, if it lands on Sunday, choose the date after that day
        if datetime.date(self.year,1,1).weekday()==6:
            holiday_dict["New Year"]=datetime.date(self.year,1,2)
        else:
            holiday_dict["New Year"]=datetime.date(self.year,1,1)

        ## Chrismas, usually 12/25, if it lands on Sunday, choose the date after that day
        if datetime.date(self.year,12,25).weekday()==5:
            holiday_dict["Chrismas"]=datetime.date(self.year,12,24)
        elif datetime.date(self.year,12,25).weekday()==6:
            holiday_dict["Chrismas"]=datetime.date(self.year,12,26)
        else:
            holiday_dict["Chrismas"]=datetime.date(self.year,12,25)

        return holiday_dict

    def num_on_off_peakdays(self,bool_east):
        """
        return the number of onpeak days and offpeak days given if the iso is from eastern power stations
        """
        if bool_east:
            weekday_threshold=5
        else:
            weekday_threshold=6 # western takes Saturday as a weekday
        fromdate = self.start_date
        todate = self.end_date
        daygenerator = [fromdate + datetime.timedelta(x ) for x in range((todate -
```

```
    fromdate).days+1)]
        #sum up all non-holiday weekdays
        num_onpeak=sum(1 for day in daygenerator if day.weekday() < weekday_threshold
    and day not in self.holiday.values())
        num_offpeak=(todate - fromdate).days+1-num_onpeak
        return num_onpeak,num_offpeak

    def is_daylight_setting(self):
        """
        check if the iso follows the daylight setting
        """
        if self.iso=="MISO":
            return False
        else:
            return True
    def daylight_dates(self):
        """
        output the start date and end date of daylight dates
        """
        cal = calendar.Calendar(0)
        month = cal.monthdatescalendar(self.year, 3)
        firstweek = month[0]
        first_sun = firstweek[-1] # the second Sunday in March
        if first_sun.month !=3:
            sec_sun=month[2][-1]
        else:
            sec_sun=month[1][-1]
        cal = calendar.Calendar(0)
        month = cal.monthdatescalendar(self.year, 11)
        firstweek = month[0]
        first_sun = firstweek[-1] # first Sunday in November
        if first_sun.month !=11:
            fir_sun=month[1][-1]
        else:
            fir_sun=month[0][-1]
        return sec_sun,fir_sun

    def hours_daylight(self,bool_east):
        """
        return the hours for daylight setting
        """
        daylight_Mar,daylight_Nov=self.daylight_dates()
        if self.peak_type == "flat":
            ## if eastern, daylight, flat
            if self.period_type == "daily":
                if daylight_Mar == self.start_date: # if the chosen date is daylight
    date in March
                    return 23
                elif daylight_Nov == self.start_date: # if the chosen date is daylight
    date in Nov
                    return 25
                else:
                    return 24
            elif self.period_type == "monthly":
                if self.start_date.month == 3: # if the chosen month contains daylight
    date in March
                    return 31*24-1
                elif self.start_date.month == 11: # if the chosen month contains
    daylight date in Nov
                    return 30*24+1
                else:
                    return self.duration*24
            elif self.period_type == "quarterly":
                    quarter=int(self.period.split("Q")[1])
                    if quarter == 1:
                        return self.duration*24-1 # if the chosen quarter contains
    daylight date in March
                    elif quarter == 4:
```

```
                        return self.duration*24+1 # if the chosen quarter contains daylight date in Nov
                    else:
                        return self.duration*24
            else:
                return self.duration*24

        ## if eastern, daylight, onpeak
        elif self.peak_type == "onpeak":
            return self.num_on_off_peakdays(bool_east)[0]*16 # daylight setting does not affect onpeak

        ## if eastern, daylight, offpeak
        elif self.peak_type == "offpeak":

            if self.period_type == "daily":
                if daylight_Mar == self.start_date: # minus 1 day if the chosen date is daylight date in March
                    return 23-self.num_on_off_peakdays(bool_east)[0]*16
                elif daylight_Nov == self.start_date: # add 1 day if the chosen date is daylight date in Nov
                    return 25-self.num_on_off_peakdays(bool_east)[0]*16
                else:
                    return 24-self.num_on_off_peakdays(bool_east)[0]*16

            elif self.period_type == "monthly":
                if self.start_date.month == 3: # minus 1 day if the chosen month contains daylight date in March
                    return 31*24-1-self.num_on_off_peakdays(bool_east)[0]*16
                elif self.start_date.month == 11: # add 1 day if the chosen month contains daylight date in Nov
                    return 30*24+1-self.num_on_off_peakdays(bool_east)[0]*16
                else:
                    return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16

            elif self.period_type == "quarterly":
                    quarter=int(self.period.split("Q")[1])
                    if quarter == 1: # minus 1 day if the chosen quarter contains daylight date in March
                        return self.duration*24-1-self.num_on_off_peakdays(bool_east)[0]*16
                    elif quarter == 4: # add 1 day if the chosen quarter contains daylight date in Nov
                        return self.duration*24+1-self.num_on_off_peakdays(bool_east)[0]*16
                    else:
                        return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16
            else:
                return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16

        elif self.peak_type == "2x16H":
            return self.num_on_off_peakdays(bool_east)[1]*16 # daylight setting does not affect H7

        elif self.peak_type == "7x8":
            if self.period_type == "daily":
                if daylight_Mar == self.start_date:# if the chosen date is daylight date in March
                    return 23-16
                elif daylight_Nov == self.start_date:# if the chosen date is daylight date in Nov
                    return 25-16
                else:
                    return 24-16

            elif self.period_type == "monthly":
                if self.start_date.month == 3:# if the chosen month contains daylight
```

```
    date in March
                    return 31*(24-16)-1
                elif self.start_date.month == 11:# if the chosen month contains
    daylight date in Nov
                    return 30*(24-16)+1
                else:
                    return self.duration*(24-16)

            elif self.period_type == "quarterly":
                    quarter=int(self.period.split("Q")[1])
                    if quarter == 1:# minus 1 day if the chosen quarter contains
    daylight date in March
                        return self.duration*(24-16)-1
                    elif quarter == 4:# add 1 day if the chosen quarter contains
    daylight date in Nov
                        return self.duration*(24-16)+1
                    else:
                        return self.duration*(24-16)
            else:
                return self.duration*(24-16)

    def hours_not_daylight(self,bool_east):
        """

        """
        if self.peak_type == "flat":
            ## if not daylight, flat
            return self.duration*24

        ## if not daylight, onpeak
        elif self.peak_type == "onpeak":
            ## daylight does not affect onpeak
            return self.num_on_off_peakdays(bool_east)[0]*16

        ## if not daylight, offpeak
        elif self.peak_type == "offpeak":
            return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16

        ## if not daylight, 2x16H
        elif self.peak_type == "2x16H":
            return self.num_on_off_peakdays(bool_east)[1]*16
        ## if not daylight, 7x8
        elif self.peak_type == "7x8":
            return self.duration*(24-16)

    def gethours(self):
        """
        get the hours given the iso, peak type and period
        """
        if self.is_eastern:
            if self.is_daylight:
                return self.hours_daylight(True)
            else:
                return self.hours_not_daylight(True)
        else:
            if self.is_daylight:
                return self.hours_daylight(False)
            else:
                return self.hours_not_daylight(False)
```

2. mmbtus is million British thermal units