

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.

```
1 import datetime
2 import calendar
3
4 class get_hours(object):
5     """
6     generate power calendar given the iso, peak type and period
7     """
8
9     def __init__(self, iso="ERCOT",peak_type="flat",period="2021A"):
10         self.iso = iso
11         self.peak_type=peak_type
12         self.period=period
13         self.year=0 # store the year of the period
14         self.period_type=self.period_tp(period) # store period type like daily, monthly
15         , quarterly and annually
16         self.start_date=self.start_dat(period)
17         self.end_date=self.end_dat(period)
18         self.is_eastern=self.is_Eastern(iso) # check if the iso belongs to eastern
19         power station
20         self.holiday=self.NERC_holiday() # create a dictionary to store all NERC
21         holiday for the year
22         self.is_daylight=self.is_daylight_setting() #check if the iso follows the
23         daylight setting
24         self.duration=(self.end_date-self.start_date).days+1 # check how many days for
25         the input
26         self.hour=self.gethours()
27
28     def period_tp(self, period):
29         """
30         get the period type
31         """
32         if period[-1]=="A":
```

```

29         return "annually"
30     elif "-" in period:
31         return "daily"
32     elif period[-2]=="Q":
33         return "quarterly"
34     else:
35         return "monthly"
36
37 def start_dat(self, period):
38     """
39     get the start date given the period
40     """
41     if self.period_type == "annually":
42         self.year=int(period[:-1])
43         return datetime.date(int(period[:-1]),1,1)
44     elif self.period_type == "daily":
45         year=int(period.split("-")[0])
46         self.year=year
47         month=int(period.split("-")[1])
48         date=int(period.split("-")[2])
49         return datetime.date(year,month,date)
50     elif self.period_type == "quarterly":
51         year=int(period.split("Q")[0])
52         self.year=year
53         quarter=int(period.split("Q")[1])
54         return datetime.date(year,quarter*3-2,1)
55     else:
56         year=int(period[:-3])
57         self.year=year
58         month_abbr=period[-3:]
59         month_num=list(calendar.month_abbr).index(month_abbr)
60         return datetime.date(year,month_num,1)
61
62 def end_dat(self, period):
63     """
64     get the end date given the period
65     """
66     if self.period_type == "annually":
67         return datetime.date(int(period[:-1]),12,31)
68     elif self.period_type == "daily":
69         year=int(period.split("-")[0])
70         month=int(period.split("-")[1])
71         date=int(period.split("-")[2])
72         return datetime.date(year,month,date)
73     elif self.period_type == "quarterly":
74         year=int(period.split("Q")[0])
75         quarter=int(period.split("Q")[1])
76         if quarter in [1,4]: # there are 31 days in the final month of quarter 1
77             return datetime.date(year,quarter*3,31)
78         else:
79             return datetime.date(year,quarter*3,30)
80     else:
81         year=int(period[:-3])
82         month_abbr=period[-3:]
83         month_num=list(calendar.month_abbr).index(month_abbr) #transfer the month
84         string to month number
85         _ ,date =calendar.monthrange(year,month_num)
86         return datetime.date(year,month_num,date)
87
88 def is_Eastern(self, iso):
89     """
90     check if the iso belongs to eastern power market
91     """
92     if iso in ["PJM","MISO","ERCOT","SPP","NYISO"]:
93         return True
94     elif iso in ["WECC","CAISO"]:
95         return False

```

```

95         else:
96             raise NameError("Please make sure the input is one of the seven iso")
97
98     def NERC_holiday(self):
99         """
100         create a dictionary to store all NERC holiday for the year
101         """
102         holiday_dict={}
103
104         ## Memorial, last Monday of May
105         cal = calendar.Calendar(firstweekday=0)
106         month = cal.monthdatescalendar(self.year, 5)
107         lastweek = month[-1]
108         monday_mem = lastweek[0]
109         holiday_dict["Memorial"]=monday_mem
110
111         ## Independence
112         holiday_dict["Independence"]=datetime.date(self.year,7,4)
113
114         ## Labor, first Monday of Sep
115         month = cal.monthdatescalendar(self.year, 9)
116         lastweek = month[0]
117         monday_la = lastweek[0]
118         if monday_la.month !=9:
119             monday_la=month[1][0]
120         holiday_dict["Labor"]=monday_la
121
122         ## Thanksgiving, fourth Thursday of Nov
123         cal = calendar.Calendar(firstweekday=0)
124         month = cal.monthdatescalendar(self.year, 11)
125         firstweek = month[0]
126         first_thrusday = firstweek[3]
127         if first_thrusday.month !=11:
128             fourth_tuesday=month[4][3]
129         else:
130             fourth_tuesday=month[3][3]
131         holiday_dict["Thanksgiving"]=fourth_tuesday
132
133         ## New year, the first date of the year, if it lands on Sunday, choose the date
134         after that day
135         if datetime.date(self.year,1,1).weekday()==6:
136             holiday_dict["New Year"]=datetime.date(self.year,1,2)
137         else:
138             holiday_dict["New Year"]=datetime.date(self.year,1,1)
139
140         ## Christmas, usually 12/25, if it lands on Sunday, choose the date after that
141         day
142         if datetime.date(self.year,12,25).weekday()==5:
143             holiday_dict["Christmas"]=datetime.date(self.year,12,24)
144         elif datetime.date(self.year,12,25).weekday()==6:
145             holiday_dict["Christmas"]=datetime.date(self.year,12,26)
146         else:
147             holiday_dict["Christmas"]=datetime.date(self.year,12,25)
148
149         return holiday_dict
150
151     def num_on_off_peakdays(self, bool_east):
152         """
153         return the number of onpeak days and offpeak days given if the iso is from
154         eastern power stations
155         """
156         if bool_east:
157             weekday_threshold=5
158         else:
159             weekday_threshold=6 # western takes Saturday as a weekday
160         fromdate = self.start_date
161         todate = self.end_date
162         daygenerator = [fromdate + datetime.timedelta(x ) for x in range((todate -

```

```

160         fromdate).days+1)]
161         #sum up all non-holiday weekdays
162         num_onpeak=sum(1 for day in daygenerator if day.weekday() < weekday_threshold
163         and day not in self.holiday.values())
164         num_offpeak=(todate - fromdate).days+1-num_onpeak
165         return num_onpeak,num_offpeak
166
167     def is_daylight_setting(self):
168         """
169         check if the iso follows the daylight setting
170         """
171         if self.iso=="MISO":
172             return False
173         else:
174             return True
175
176     def daylight_dates(self):
177         """
178         output the start date and end date of daylight dates
179         """
180         cal = calendar.Calendar(0)
181         month = cal.monthdatescalendar(self.year, 3)
182         firstweek = month[0]
183         first_sun = firstweek[-1] # the second Sunday in March
184         if first_sun.month !=3:
185             sec_sun=month[2][-1]
186         else:
187             sec_sun=month[1][-1]
188         cal = calendar.Calendar(0)
189         month = cal.monthdatescalendar(self.year, 11)
190         firstweek = month[0]
191         first_sun = firstweek[-1] # first Sunday in November
192         if first_sun.month !=11:
193             fir_sun=month[1][-1]
194         else:
195             fir_sun=month[0][-1]
196         return sec_sun,fir_sun
197
198     def hours_daylight(self,bool_east):
199         """
200         return the hours for daylight setting
201         """
202         daylight_Mar,daylight_Nov=self.daylight_dates()
203         if self.peak_type == "flat":
204             ## if eastern, daylight, flat
205             if self.period_type == "daily":
206                 if daylight_Mar == self.start_date: # if the chosen date is daylight
207                     date in March
208                     return 23
209                 elif daylight_Nov == self.start_date: # if the chosen date is daylight
210                     date in Nov
211                     return 25
212             else:
213                 return 24
214             elif self.period_type == "monthly":
215                 if self.start_date.month == 3: # if the chosen month contains daylight
216                     date in March
217                     return 31*24-1
218                 elif self.start_date.month == 11: # if the chosen month contains
219                     daylight date in Nov
220                     return 30*24+1
221             else:
222                 return self.duration*24
223             elif self.period_type == "quarterly":
224                 quarter=int(self.period.split("Q")[1])
225                 if quarter == 1:
226                     return self.duration*24-1 # if the chosen quarter contains
227                     daylight date in March
228                 elif quarter == 4:

```

```

221         return self.duration*24+1 # if the chosen quarter contains
daylight date in Nov
222     else:
223         return self.duration*24
224     else:
225         return self.duration*24
226
227     ## if eastern, daylight, onpeak
228     elif self.peak_type == "onpeak":
229         return self.num_on_off_peakdays(bool_east)[0]*16 # daylight setting does
not affect onpeak
230
231     ## if eastern, daylight, offpeak
232     elif self.peak_type == "offpeak":
233
234         if self.period_type == "daily":
235             if daylight_Mar == self.start_date: # minus 1 day if the chosen date is
daylight date in March
236                 return 23-self.num_on_off_peakdays(bool_east)[0]*16
237             elif daylight_Nov == self.start_date: # add 1 day if the chosen date is
daylight date in Nov
238                 return 25-self.num_on_off_peakdays(bool_east)[0]*16
239             else:
240                 return 24-self.num_on_off_peakdays(bool_east)[0]*16
241
242         elif self.period_type == "monthly":
243             if self.start_date.month == 3: # minus 1 day if the chosen month
contains daylight date in March
244                 return 31*24-1-self.num_on_off_peakdays(bool_east)[0]*16
245             elif self.start_date.month == 11: # add 1 day if the chosen month
contains daylight date in Nov
246                 return 30*24+1-self.num_on_off_peakdays(bool_east)[0]*16
247             else:
248                 return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16
249
250         elif self.period_type == "quarterly":
251             quarter=int(self.period.split("Q")[1])
252             if quarter == 1: # minus 1 day if the chosen quarter contains
daylight date in March
253                 return self.duration*24-1-self.num_on_off_peakdays(bool_east)
[0]*16
254             elif quarter == 4: # add 1 day if the chosen quarter contains
daylight date in Nov
255                 return self.duration*24+1-self.num_on_off_peakdays(bool_east)
[0]*16
256             else:
257                 return self.duration*24-self.num_on_off_peakdays(bool_east)
[0]*16
258         else:
259             return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16
260
261     elif self.peak_type == "2x16H":
262         return self.num_on_off_peakdays(bool_east)[1]*16 # daylight setting does
not affect H7
263
264     elif self.peak_type == "7x8":
265         if self.period_type == "daily":
266             if daylight_Mar == self.start_date:# if the chosen date is daylight
date in March
267                 return 23-16
268             elif daylight_Nov == self.start_date:# if the chosen date is daylight
date in Nov
269                 return 25-16
270             else:
271                 return 24-16
272
273         elif self.period_type == "monthly":
274             if self.start_date.month == 3:# if the chosen month contains daylight

```

```

275         date in March
276             return 31*(24-16)-1
277         elif self.start_date.month == 11:# if the chosen month contains
278             daylight date in Nov
279             return 30*(24-16)+1
280             else:
281                 return self.duration*(24-16)
282
283         elif self.period_type == "quarterly":
284             quarter=int(self.period.split("Q")[1])
285             if quarter == 1:# minus 1 day if the chosen quarter contains
286                 daylight date in March
287                 return self.duration*(24-16)-1
288             elif quarter == 4:# add 1 day if the chosen quarter contains
289                 daylight date in Nov
290                 return self.duration*(24-16)+1
291             else:
292                 return self.duration*(24-16)
293
294         else:
295             return self.duration*(24-16)
296
297     def hours_not_daylight(self, bool_east):
298         """
299
300         """
301         if self.peak_type == "flat":
302             ## if not daylight, flat
303             return self.duration*24
304
305         ## if not daylight, onpeak
306         elif self.peak_type == "onpeak":
307             ## daylight does not affect onpeak
308             return self.num_on_off_peakdays(bool_east)[0]*16
309
310         ## if not daylight, offpeak
311         elif self.peak_type == "offpeak":
312             return self.duration*24-self.num_on_off_peakdays(bool_east)[0]*16
313
314         ## if not daylight, 2x16H
315         elif self.peak_type == "2x16H":
316             return self.num_on_off_peakdays(bool_east)[1]*16
317         ## if not daylight, 7x8
318         elif self.peak_type == "7x8":
319             return self.duration*(24-16)
320
321     def gethours(self):
322         """
323         get the hours given the iso, peak type and period
324         """
325         if self.is_eastern:
326             if self.is_daylight:
327                 return self.hours_daylight(True)
328             else:
329                 return self.hours_not_daylight(True)
330         else:
331             if self.is_daylight:
332                 return self.hours_daylight(False)
333             else:
334                 return self.hours_not_daylight(False)

```

2. mmbtus is million British thermal units