General idea:

1. The main function is called ‘get_hours’. According to the input format of period, run sub-function correspondingly. For example, if the period format is ‘2018A’, run the sub-function ‘get_yearly’.

2. For the sub-function, e.g., get_yearly, the logic is that we generate a list of dates inside the time range of that year and run ‘get_daily’ for each date. Finally, sum the results.

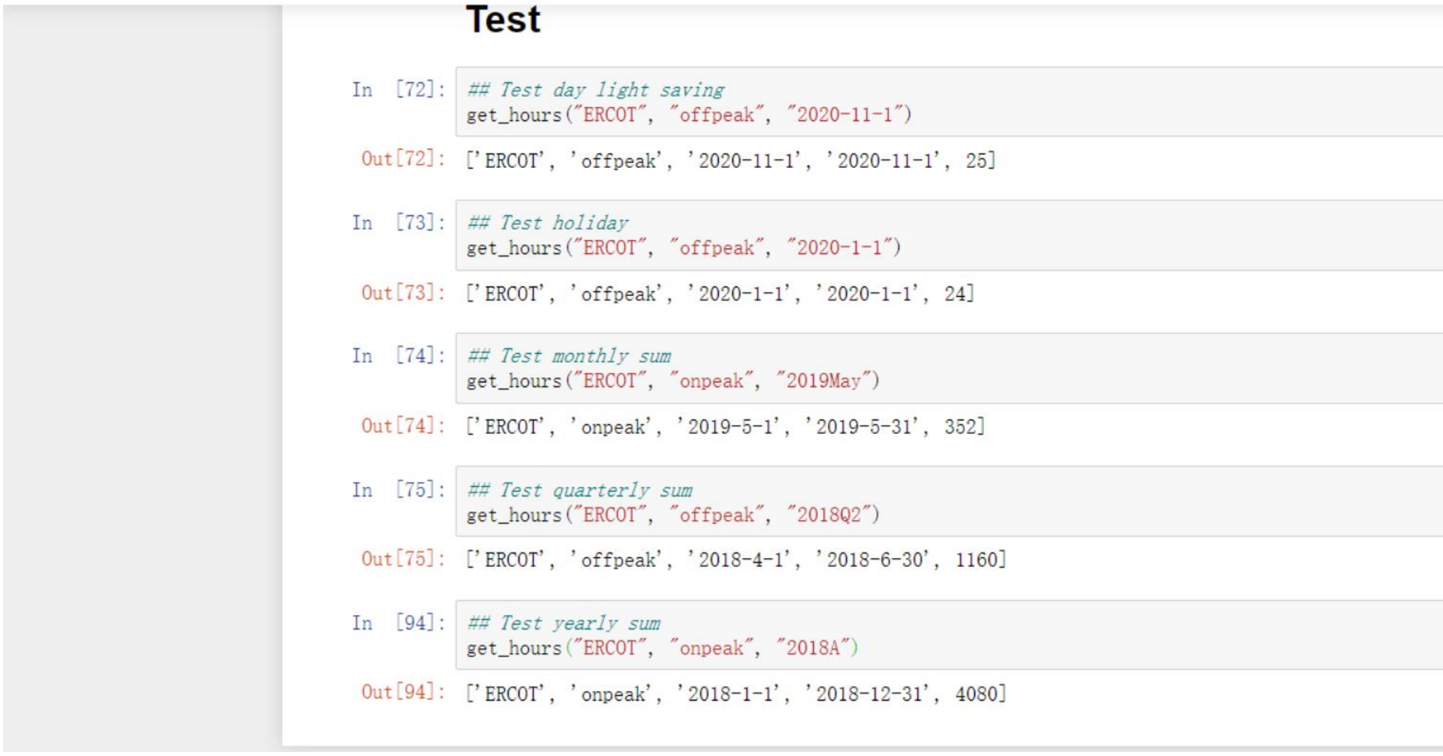
Some nuances:

1. For holidays, in python, we do not have a built-in function perfect for NERC holidays. Thus, a function is defined for detecting the NERC holidays.

2. For the starting date and ending date of day light saving, they are different for different time zones. At this stage, I just select the time-zone of US central. Further update should be changing the time-zone according to iso.

3. Considering the limitation of time, I did not ensure the robustness of the function, which means that I am assuming the input format is always correct.

Test results:



Test

```
In  [72]: ## Test day light saving
          get_hours("ERCOT", "offpeak", "2020-11-1")

Out[72]: ['ERCOT', 'offpeak', '2020-11-1', '2020-11-1', 25]

In  [73]: ## Test holiday
          get_hours("ERCOT", "offpeak", "2020-1-1")

Out[73]: ['ERCOT', 'offpeak', '2020-1-1', '2020-1-1', 24]

In  [74]: ## Test monthly sum
          get_hours("ERCOT", "onpeak", "2019May")

Out[74]: ['ERCOT', 'onpeak', '2019-5-1', '2019-5-31', 352]

In  [75]: ## Test quarterly sum
          get_hours("ERCOT", "offpeak", "2018Q2")

Out[75]: ['ERCOT', 'offpeak', '2018-4-1', '2018-6-30', 1160]

In  [94]: ## Test yearly sum
          get_hours("ERCOT", "onpeak", "2018A")

Out[94]: ['ERCOT', 'onpeak', '2018-1-1', '2018-12-31', 4080]
```