Assignment 1: Power Calendar function

Function "get_hours(iso, peak_type, period)"

Description

Calculates the number of hours matching specified peak type across different Independent System Operators (ISOs). It supports filtering by peak type and adjusts for holidays and weekends as applicable to each ISO's operational rules.

Parameters

- iso (str): The identifier for the ISO (e.g., 'PJM', 'NYISO', 'CAISO').
- peak_type (str): Type of peak hours ('onpeak', 'offpeak', 'flat', '2x16h', '1x16h', '7x8').
- period (str): The period for the calculation. Supported formats include daily (e.g., '2018-02-03'), quarterly (e.g., '2018Q2'), monthly (e.g., '2018Mar'), and annually (e.g., '2018A').

Returns

- dict: A dictionary containing:
- iso: The ISO processed.
- peak_type: The peak type considered.
- startdate: The formatted start date of the period.
- enddate: The formatted end date of the period.
- num_hours: The number of hours that match the specified criteria.

Function "get_nerc_holidays(start_date, end_date, tz=None)"

Description

This function retrieves the NERC holidays for a specified date range within the United States. It adjusts holidays that fall on a Sunday to the next day (Monday), in accordance with NERC regulations. This function serves function "get_hours(iso, peak_type, period)" to get NERC holidays.

Parameters

- start_date (str or datetime): The starting date of the period for which to retrieve holidays.
- end_date (str or datetime): The ending date of the period for which to retrieve holidays.
- tz (str, optional): The timezone for each ISO to manage daylight saving setting. Default is None.

Returns

- pandas.DatetimeIndex: An index of datetime objects representing NERC holidays within the given date range, localized to the specified timezone.

In [1]:
```
import pandas as pd
import holidays
import pytz

def get_nerc_holidays(start_date, end_date, tz=None):
    # Format start_date and end_date to datetime
    start_date = pd.to_datetime(start_date).tz_localize(tz)
    end_date = pd.to_datetime(end_date).tz_localize(tz)

    # Create an object for U.S. public holidays
    us_holidays = holidays.US(years=range(start_date.year, end_date.year + 1))

    # Define NERC holidays based on fixed and observed holidays
    nerc_holiday_names = {
        "New Year's Day",
        "Memorial Day",
        "Independence Day",
        "Labor Day",
        "Thanksgiving",
        "Christmas Day"
    }

    adjusted_holidays = {}
    # Adjust holidays that fall on Sunday to the next day
    # e.g #2023 New York Day on 2023/1/2; 2022 Chrismas on 12/26/2022
    for date, name in us_holidays.items():
        if name in nerc_holiday_names:
            current_date = pd.Timestamp(date)
            # Check if the holiday falls on a Sunday
            if current_date.weekday() == 6:  # Sunday
                # Shift to the next day (Monday)
                observed_date = current_date + pd.Timedelta(days=1)
                adjusted_holidays[observed_date] = name
            else:
                adjusted_holidays[current_date] = name

    # Filter out the holidays within the given date range
    holiday_dates = [
        pd.Timestamp(date).tz_localize(None).tz_localize(tz)
        for date, name in adjusted_holidays.items()
        if name in nerc_holiday_names
    ]
    holiday_dates = [
        date for date in holiday_dates if date >= start_date and date <= end_date
    ]

    return pd.DatetimeIndex(holiday_dates)
```

In [2]:
```
def get_hours(iso, peak_type, period):

    if '-' in period:
        # Daily format(e.g."2018-02-03")
        start_date = pd.to_datetime(period)
        end_date = start_date
    elif 'Q' in period:
        # Quarterly format(e.g., "2018Q2")
        start_date = pd.Period(period, freq='Q').start_time
        end_date = pd.Period(period, freq='Q').end_time
    elif len(period) == 7:
        # Monthly format(e.g., "2018Mar")
        start_date = pd.to_datetime(period, format='%Y%b')
        end_date = start_date + pd.offsets.MonthEnd(1)
    elif period.endswith('A'):
        # Annually format(e.g., "2018A")
        year = pd.to_datetime(period[:-1] + '-01-01')
        start_date = year
        end_date = start_date + pd.offsets.YearEnd(1)

    # Generate range of dates depend on different iso
    if  iso.upper() == 'PJM' or iso == 'NYISO':
        # PJM and NYISO use US Eastern Time year round
        all_dates = pd.date_range(start=start_date,
                                  end=end_date + pd.Timedelta(days=1),
                                  freq='H',tz = pytz.timezone('US/Eastern'))
        holidays = get_nerc_holidays(start_date, end_date, tz='US/Eastern')

    elif iso.upper() == 'SPP' or iso.upper() == 'ERCOT':
        #SPP and ERCOT use US Central Standard
        all_dates = pd.date_range(start=start_date, end=end_date + pd.Timedelta(days=1),
                                  freq='H',tz=pytz.timezone('US/Central'))
        holidays = get_nerc_holidays(start_date, end_date, tz='US/Central')


    elif iso.upper() ==  'CAISO' or iso.upper() == 'WECC':
        all_dates = pd.date_range(start=start_date,
                                  end=end_date + pd.Timedelta(days=1),
                                  freq='H',tz=pytz.timezone('US/Pacific'))
        holidays = get_nerc_holidays(start_date, end_date, tz='US/Pacific')

    elif iso.upper() == 'MISO': # MISO does not have the daylight-saving setting
        all_dates = pd.date_range(start=start_date,
                                  end=end_date + pd.Timedelta(days=1), freq='H')
        holidays = get_nerc_holidays(start_date, end_date)


    if 'Q' in period:
        all_dates = all_dates[:-24]
    else:
        all_dates = all_dates[:-1]


    if peak_type.lower() == "onpeak":   # non-NERC holiday weekday from HE7 to HE22
        if iso.upper() ==  'CAISO' or iso.upper() == 'WECC':
            # Heavy load for CAISO and WECCHE7 to HE22 from Monday to Saturday
            peakdays = all_dates[(all_dates.weekday < 6)  & ~(all_dates.normalize().isin(holidays))]

        else:  # HE7 to HE22 from Monday to Friday
            peakdays = all_dates[(all_dates.weekday < 5)  & ~(all_dates.normalize().isin(holidays))]
        valid_hours = range(6, 22)     # (HE7 to HE22)
        valid_dates = peakdays[peakdays.hour.isin(valid_hours)]

    elif peak_type.lower() == "flat": # HE1 to HE24 every day
        valid_dates = all_dates

    elif peak_type.lower() == "offpeak":   # flat hour - peak hour
```

```
        if iso.upper() ==  'CAISO' or iso.upper() == 'WECC':
            peakdays = all_dates[(all_dates.weekday < 6)  & ~(all_dates.normalize().isin(holidays))]

        else:
            peakdays = all_dates[(all_dates.weekday < 5)  & ~(all_dates.normalize().isin(holidays))]
        peak_dates = peakdays[peakdays.hour.isin(range(6, 22))]
        valid_dates = all_dates[~(all_dates.isin(peak_dates))]


    elif peak_type.lower() == "2x16h":    # HE7 to HE22 for the weekend and the NERC holiday

        valid_days = all_dates[(all_dates.weekday >= 5) | (all_dates.normalize().isin(holidays))]
        # weekend and holidays
        valid_hours = range(6, 22)
        valid_dates = valid_days[valid_days.hour.isin(valid_hours)]

    elif peak_type.lower() == "1x16h":    # CAISO and WECC has 1x16H instead
        valid_days = all_dates[(all_dates.weekday > 5) | (all_dates.normalize().isin(holidays))]
        valid_hours = range(6, 22)
        valid_dates = valid_days[valid_days.hour.isin(valid_hours)]


    elif peak_type.lower() == "7x8":   # Non HE7 to HE22 through the week
        valid_days = all_dates
        valid_hours = list(range(0, 6)) + [22, 23]
        valid_dates = valid_days[valid_days.hour.isin(valid_hours)]


    # Count the valid hours
    num_hours = len(valid_dates)

    res = {
        'iso': iso.upper(),
        'peak_type': peak_type.upper(),
        'startdate': start_date.strftime('%Y-%m-%d'),
        'enddate': end_date.strftime('%Y-%m-%d'),
        'num_hours': num_hours
    }
    return res
```

Test the function

This code tests the get_hours function across various ISOs for November 2023.

- PJM and ERCOT: Both ISOs report 336 onpeak hours and 386 offpeak hours. The additional offpeak hour is attributed to the end of daylight saving time, which adds an extra hour in November.
- MISO: Shows 385 offpeak hours. Unlike PJM and ERCOT, MISO operates on standard time year-round and does not adjust for daylight savings.
- WECC and CAISO: These ISOs both have 400 onpeak and 322 offpeak hours. The higher number of onpeak hours results from their specific peak hour settings: HE7 to HE22 from Monday to Saturday, excluding NERC holidays. This is broader than other ISOs, which typically count HE7 to HE22 from Monday to Friday, excluding NERC holidays.

In [3]:
```
#test the function, use 2023 Nov to see if the daylight saving setting works
iso_list = ['PJM', 'MISO', 'ERCOT', 'WECC', 'CAISO']
peak_type_list = ['onpeak','offpeak']
for iso in iso_list:
    print(iso)
    for peak_type in peak_type_list:
        res = get_hours(iso, peak_type, '2023Nov')
        print(res)
    print("-------------------------------------------------------------------------------------")
```

```
PJM
{'iso': 'PJM', 'peak_type': 'ONPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 336}
{'iso': 'PJM', 'peak_type': 'OFFPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 385}
--------------------------------------------------------------------------------------
MISO
{'iso': 'MISO', 'peak_type': 'ONPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 336}
{'iso': 'MISO', 'peak_type': 'OFFPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 384}
--------------------------------------------------------------------------------------
ERCOT
{'iso': 'ERCOT', 'peak_type': 'ONPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 336}
{'iso': 'ERCOT', 'peak_type': 'OFFPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 385}
--------------------------------------------------------------------------------------
WECC
{'iso': 'WECC', 'peak_type': 'ONPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 400}
{'iso': 'WECC', 'peak_type': 'OFFPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 321}
--------------------------------------------------------------------------------------
CAISO
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 400}
{'iso': 'CAISO', 'peak_type': 'OFFPEAK', 'startdate': '2023-11-01', 'enddate': '2023-11-30', 'num_hours': 321}
--------------------------------------------------------------------------------------
```

This following code tests the the get_hours function for different ISOs over the fourth quarter of 2023 (October to December).

- PJM and ERCOT: Both ISOs report 1008 onpeak hours and 1201 offpeak hours for the quarter. The additional offpeak hours arise due to the end of daylight saving time in November.
- MISO: Reflects 1008 onpeak and 1200 offpeak hours. The slight difference in offpeak hours compared to PJM and ERCOT (1201 vs. 1200) is because MISO does not observe daylight saving changes.
- WECC and CAISO: Both ISOs show 1216 onpeak and 993 offpeak hours. The increased onpeak hours result from their extended peak hours definition, which includes HE7 to HE22 from Monday to Saturday, excluding NERC holidays. This broader definition of peak hours in WECC and CAISO results in fewer offpeak hours compared to other ISOs that typically observe peak hours from Monday to Friday.

In [4]:
```python
iso_list = ['PJM', 'MISO', 'ERCOT', 'WECC', 'CAISO']
peak_type_list = ['onpeak','offpeak']
for iso in iso_list:
    print(iso)
    for peak_type in peak_type_list:
        res = get_hours(iso, peak_type, '2023Q4')
        print(res)
    print("--------------------------------------------------------------------------------------")
```

```
PJM
{'iso': 'PJM', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1008}
{'iso': 'PJM', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1201}
--------------------------------------------------------------------------------------
MISO
{'iso': 'MISO', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1008}
{'iso': 'MISO', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1200}
--------------------------------------------------------------------------------------
ERCOT
{'iso': 'ERCOT', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1008}
{'iso': 'ERCOT', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1201}
--------------------------------------------------------------------------------------
WECC
{'iso': 'WECC', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1216}
{'iso': 'WECC', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 993}
--------------------------------------------------------------------------------------
CAISO
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1216}
{'iso': 'CAISO', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 993}
--------------------------------------------------------------------------------------
```

In [5]:
```python
# The outputs of the get_hours function are the same as those shown on the given website,
# producing accurate results as expected.

peak_type_list = ['onpeak','offpeak','1X16H']
for peak in peak_type_list:

    res = get_hours('CAISO', peak, '2023Q4')
    print(res)
```

```
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 1216}
{'iso': 'CAISO', 'peak_type': 'OFFPEAK', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 993}
{'iso': 'CAISO', 'peak_type': '1X16H', 'startdate': '2023-10-01', 'enddate': '2023-12-31', 'num_hours': 256}
```

In [6]:
```python
year_list = ['2018A','2019A','2020A','2021A','2022A','2023A']
for year in year_list:
    res = get_hours('CAISO', 'onpeak', year)
    print(res)
```

```
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2018-01-01', 'enddate': '2018-12-31', 'num_hours': 4912}
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2019-01-01', 'enddate': '2019-12-31', 'num_hours': 4912}
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2020-01-01', 'enddate': '2020-12-31', 'num_hours': 4928}
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2021-01-01', 'enddate': '2021-12-31', 'num_hours': 4912}
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2022-01-01', 'enddate': '2022-12-31', 'num_hours': 4912}
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2023-01-01', 'enddate': '2023-12-31', 'num_hours': 4896}
```

Test the onpeak hours for January 2, 2023, which is a NERC-recognized New Year holiday. Since the traditional New Year's Day falls on a Sunday, Monday is observed as the holiday, in accordance with NERC regulations. The expected number of onpeak hours at CAISO on January 2, 2023, should be 0.

In [7]:
```
res = get_hours('CAISO', 'onpeak', '2023-1-2')
print(res)
```

```
{'iso': 'CAISO', 'peak_type': 'ONPEAK', 'startdate': '2023-01-02', 'enddate': '2023-01-02', 'num_hours': 0}
```

Note:for assignment1, I made an independent python file named power_calendar.py under the "code" folder

In [ ]:

In [ ]:

In [ ]: