Assignment 1:

Shuxiang Sui

Objective: write R/Python function which returns number of hours by iso/peak.type/period

In power market, the industry defines certain hour of each day to peak type for block trading, so we need to calculate correctly how many hours belongs to certain block. Each ISO has a little different definition of it.

Note: don't scrape data from the reference link. It's for reference only. You shall learn/understand the logic and calculate without access to any website.

```python
import holidays

us_holidays = holidays.US(years=2019, prov='NERC')

print("US NERC Holidays:")
for date, name in sorted(us_holidays.items()):
    print(date, name)
```

```
US NERC Holidays:
2019-01-01 New Year's Day
2019-01-21 Martin Luther King Jr. Day
2019-02-18 Washington's Birthday
2019-05-27 Memorial Day
2019-07-04 Independence Day
2019-09-02 Labor Day
2019-10-14 Columbus Day
2019-11-11 Veterans Day
2019-11-28 Thanksgiving
2019-12-25 Christmas Day
```

Firstly, I gather the data for US holiday and I found out that the US holidays are more than NERC holidays, therefore I set the place to manually input the holiday date for this.

At the beginning of this function, I defined 3 inputs and count period length in order to match the 4 types of Period characters, and respectively produce the answer for daily, monthly, quarterly, and annually.

```
#Annually
if period_length == 5:
    start_date = datetime(year, 1, 1)
    end_date = datetime(year, 12, 31)

#Quarterly
elif period_length == 6:
    quarter = int(period[5])

    start_month = 3 * (quarter - 1) + 1
    start_date = datetime(year, start_month, 1)

    end_month = start_month + 2
    end_date = (datetime(year, end_month + 1, 1) - timedelta(days=1))

#Monthly
elif period_length == 7:
    month = datetime.strptime(period[4:], "%b").month
    start_date = datetime(year, month, 1)
    end_date = (datetime(year, month + 1, 1) - timedelta(days=1))

# Daily
elif period_length > 7 and period[-1].isdigit():
    start_date = datetime.strptime(period, "%Y-%m-%d") #phrase to datetime format
    end_date = start_date
else:
    raise ValueError("This is a Wrong Input,\n Input instruction: \n“2018-2-3” as a daily, “2018Mar” as a monthly, “2018Q
```

For here, I use the length method to identify the input and give the output respectively.

When Length =5, it is Annually,

When Length = 6, it is Quarterly,

When Length = 7, it is Monthly,

When Length >7 but ending in number, it will be Daily.

For any Wrong input in any step, you will see the feedback:

"This is a Wrong Input,

Input instruction:

“2018-2-3” as a daily, “2018Mar” as a monthly, “2018Q2” as a quarterly, “2018A” as an annually."

```
# Process peak type
## peak.type (character): one of onpeak/offpeak/flat/2x16H/7x8

#daylight saving indicator
include = 0
days = pd.date_range(start=start_date, end=end_date, freq='D')

#Flat Peak for all day
if peak_type == "flat":
    include = 1
    hours = len(days) * 24

#everyday of a week, for 8 hours
elif peak_type == "7x8":
    include = 1
    hours = len(days) * 8

else:
    weekdays = days.weekday
    year_holidays = [datetime.strptime(date_str, "%Y-%m-%d") for date_str in ["2019-01-01", "2019-05-27", "2019-07-04",
    holidays = [d for d in year_holidays if start_date <= d <= end_date]
    non_holidays = [d for d in days if d not in holidays]

    # Process ISO
    if iso in ["PJMISO", "MISO", "ERCOT", "SPPISO", "NYISO"]:  # Eastern market
        weekend = [5, 6]
    elif iso in ["WECC", "CAISO"]:  # Western market
        weekend = [5]
    else:
        raise ValueError("Wrong input of iso")

    non_holiday_weekdays = [d for d in non_holidays if d.weekday() not in weekend]
```

Since MISO does not have the daylight-saving setting, the rest have, I add the indicator before calculating the total hours.

The weekday and Weekend difference between East and West coast are also considered and there will be an ValueError raised as the reminder if anything goes wrong.

There are Flat Peak, 7x8, Weekends(2*16), and Holiday, for Holidays, I set it to manually input, I could also apply API method to get holiday dates but this should work offline too.

```
    if peak_type == "onpeak":
        hours = len(non_holiday_weekdays) * 16

    elif peak_type == "offpeak":
        include = 1
        hours = len(days) * 24 - len(non_holiday_weekdays) * 16

    elif peak_type == "2x16H":
        weekend_holidays = list(set(holidays).union(set(days[weekdays.isin(weekend)])))
        hours = len(set(weekend_holidays)) * 16

#Day light saving
if iso != "MISO" and include == 1:  # only MISO not participate
    march_dates = pd.date_range(start=datetime(year, 3, 1), end=datetime(year, 3, 31))
    march_sundays = [d for d in march_dates if d.weekday() == 6]
    # Add daylight-saving time

# Return the result
result = {
    "iso": iso,
    "peak_type": peak_type,
    "start_date": start_date,
    "end_date": end_date,
    "num_hour": hours
}
return result
```

Besides, I also need to avoid the conflict between the Holiday and others, and the danger of counting it twice, the new list is created.

Then, we need to calculate the hours base on previous result of peak type, by providing the space for daylight adjustment, the calculator could also easily adjust the changes in Daylight saving event.

At the End, I tested the result using:

```
In [5]: try1 = get_hours("ERCOT", "onpeak", "2019May")
        try1

Out[5]: {'iso': 'ERCOT',
         'peak_type': 'onpeak',
         'start_date': datetime.datetime(2019, 5, 1, 0, 0),
         'end_date': datetime.datetime(2019, 5, 31, 0, 0),
         'num_hour': 352}
```

Lucky my calculator passes the test.