In [ ]:
```
import os
import numpy as np
import pandas as pd
import pvlib #For simulation of PV performance
import matplotlib.pyplot as plt
import pathlib
from pvlib.pvsystem import PVSystem
```

In [ ]:

Irradiance data extracted from solcast with 5 minutes interval for 04-04-2023

For Solar Plant of Clodfelter - Kennewick WA 14.640kW

In [ ]:
```
#Reading the dataset

pv = r"D:\Thesis KAU\Clodfelter - Kennewick WA 14.640kW\04-04-2023.xlsx"

#raw string notation is used to indicate the file path of the csv file

pv_data = pd.read_excel(pv)

pv_data.index = pd.to_datetime(pv_data[['Year', 'Month', 'Day', 'Hour', 'Minute']]
# index is used to identify and access the rows of the DataFrame
# convert a column of data in a DataFrame to a datetime data type

pv_data = pv_data.tz_localize('UTC')
#the time zone is set to 'UTC', which stands for Coordinated Universal Time
```

In [ ]:
```
pv_data.head(5) #getting the first five rows of dataset
```

In [ ]:
```
#checking the number of rows and columns in dataset

print("Number of rows:", len(pv_data))
print("Number of columns:", len(pv_data.columns))
```

In [ ]:
```
pv_data.keys() #data features that we have in our dataset
```

Data Exploration

We are going to focus on the features that are important for PV modeling among the multiple columns present in the dataset.

In [ ]:
```
data = pv_data[['GHI', 'DHI', 'DNI', 'Tamb', 'WindVel']]
data.head(5)
```

Plotting irradiance data for better understanding of data set

In [ ]:
```
#we are plotting the irradiance data for the first week of july

dec=data.loc['2023-04-04']
dec[['GHI', 'DHI', 'DNI']].plot(figsize=(14,5))
plt.ylabel('Irradiance [W/m$^2$]');
plt.xlabel('Time');

#from below graph we can conclude that DNI is highest in most of the days,
#due to seasonal variation GHI is more than DNI
```

Irradiance pattern for a single day.

In [ ]:

In [ ]:
```
#irradiance pattern in summer

single_day_s=data.loc['2023-04-04']
single_day_s[['GHI', 'DHI', 'DNI']].plot(figsize=(10,5))
plt.ylabel('Irradiance [W/m$^2$]');
plt.xlabel('Time')

#In winter, the sun is at a lower angle in the sky, which increases the
#amount of atmospheric scattering and increases the amount of diffuse radiation rec
```

In [ ]:

In [ ]:

Temperature trend from January to August.

In [ ]:
```
temp=data.loc['2023-04-04']['Tamb'].plot(figsize=(14,5))
plt.ylabel('Ambient Temperature [°C]');
plt.xlabel('04th April')
```

Wind velocity trend in the month of June

In [ ]:
```
wind=data.loc['2023-04-04']['WindVel'].plot(figsize=(14,5))
plt.ylabel('Wind Speed [m/s]');
plt.xlabel('04th April');
```

POA Irradiance

Plane of array irradiance refers to the amount of solar radiation (i.e. sunlight) that is received by the surface of a solar panel or array of solar panels. It is typically measured in units of watts per square meter (W/m^2) and is affected by factors such as the angle of the sun, the amount of cloud cover, and the location of the solar panel. The optimal plane of array irradiance for a solar panel is when it is perpendicular to the sun's rays, as this allows the panel to receive the most amount of sunlight possible.

Transforming the GHI,DHI and DNI in to the main POA component because main solar radiation is POA that is imapcting the specific surface that is solar panel

We need to calculate solar positions first to get the POA irradiance

In [ ]:
```
times = data.index - pd.Timedelta('5min')
times
```

In [ ]:
```
#Now we will transform the three irradiance in to the POA irradiance which is most
#Transposition  - It is the calculation of incident irradiance on a tilted plane fr
#Calculating the POA from GHI, DNI and DHI
#Below are the geo point for GEC

location = pvlib.location.Location(latitude=46.175213, longitude=-119.262776)

# Time interval in the dataset is 1 hour, data gives us the average weather condit
# 09AM to 10AM interval is labeled 10.  We will calculate solar position in
# the middle of the interval (09:30), so we subtract 30 minutes

times = data.index - pd.Timedelta('5min')

# function helps to calculate the solar zenith, azimuth, etc. at this location.

solar_position = location.get_solarposition(times)

# shifting the index back to line up with the dataset:

solar_position.index = solar_position.index + pd.Timedelta('5min')
solar_position.head()
```

Zenith (How close the sun is to overhead) and Azimuth (what direction along the horizon the sun is, like panel azimuth).

Apparent_zenith is with effect of atmosphere and zenith is without effect of atmosphere

In [ ]:
```
#Our PV array is facing south (azimuth = 180) with fixed tilt angle of 40 degree

#There are multiple models for calculation we are going to use the default model=is
#some complex models that requires more weather inputs. Perez and Hay Davies models

poa_irr = pvlib.irradiance.get_total_irradiance(surface_tilt=45,surface_azimuth=18
                                                ghi=data['GHI'],
                                                dhi=data['DHI'],
                                                solar_zenith=solar_position['appar
                                                solar_azimuth=solar_position['azim
                                                model='isotropic')
poa_irr[90:120]


# Below are the values of POA for a single day
# In below dataframe we got the POA components of all irradiance
```

In [ ]:

Effective irradiance

Plane of array (POA) irradiance is the amount of solar radiation that falls on a surface perpendicular to the sun's rays. Effective irradiance, also known as plane of cell (POC) irradiance, is the amount of solar radiation that falls on a surface at an angle, taking into account the effect of shading and reflections from surrounding surfaces. The POA irradiance is typically higher than the POC irradiance, because the POC irradiance takes

into account the reduction in solar radiation due to shading and reflections.

An incident angle modifier (IAM) is a model used to adjust the amount of solar irradiance that reaches a solar panel based on the angle at which sunlight strikes the panel. The IAM takes into account the angle of incidence (AOI) of the sun's rays on the panel and modifies the irradiance value based on the specific characteristics of the panel.

In [ ]:
```
#we have to calculate the effective irradiance also for the model.

#AOI is the angle at which sunlight strikes a solar panel and
#its effect on the panel's power output.


aoi=pvlib.irradiance.aoi(surface_tilt=45,surface_azimuth=180,
                         solar_zenith=solar_position['apparent_zenith'],
                         solar_azimuth=solar_position['azimuth'])

#Determine the incidence angle modifier using the ASHRAE transmission model.

iam=pvlib.iam.ashrae(aoi)

#effective irradiance

effective_irradiance=poa_irr['poa_direct'] * iam + poa_irr['poa_diffuse']

effective_irradiance['2023-04-04 10:00:00+00:00']
```

In [ ]:
```
#We will see monthly irradiance to observe the difference between the the amount o
#flat panel (GHI) and that of a tilted panel (POA):
```

In [ ]:
```
#Visualizing the trend for a single day in winter

day=df.loc['2023-04-04']
plt.figure(figsize=(10,6))
plt.plot(day)
plt.ylabel('Irradiance [W/m$^2$]');
plt.xlabel('Hours of the day')
plt.legend(['ghi','poa'])
```

Estimating the PV cell temp

Sandia Array Performance Model (SAPM) estimate cell temperature from ambient conditions. The SAPM thermal model takes POA irradiance, ambient temperature, and wind speed as weather inputs

In [ ]:
```
#sandia Array performance model(sapm) is used to calculate the cell temperature.
#We are considering here open rack structure with polymer backsheet

all_parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_
cell_temperature = pvlib.temperature.sapm_cell(poa_irr['poa_global'],data['Tamb'],
                                               data['WindVel'],**all_parameters)
cell_temperature.head(288)
```

Comapring the cell temp with the ambient temp for the first week

In [ ]:
```
plt.figure(figsize=(10,6))
data['Tamb'].head(288).plot()
cell_temperature.head(288).plot()
```

```
plt.legend(['Tamb', 'Cell Temperature'])
plt.ylabel('Temperature [°C]');
plt.xlabel('Time')
```

In the above graph we see that cell temp is more than ambient temperature because of the heat generated by the PV effect.

In [ ]:

PV module and inverter selection from database

In [ ]:
```
#importing the modules and inverter from from database

cec_modules = pvlib.pvsystem.retrieve_sam('CECMod') #database modules
cec_inverters = pvlib.pvsystem.retrieve_sam('cecinverter') #database inverter

PV_module_cec = cec_modules['CSUN_Eurasia_Energy_Systems_Industry_and_Trade_CSUN30
```

In [ ]:
```
PV_module_cec
```

In [ ]:

In [ ]:

Calculates five parameter values for the single diode equation at effective irradiance and cell temperature using the CEC model

In [ ]:
```
#The five values are returned by calcparams_cec can be used by singlediode to calcu


cec_params=pvlib.pvsystem.calcparams_cec(effective_irradiance=effective_irradiance,
                                         temp_cell=cell_temperature,
                                         alpha_sc=PV_module_cec.alpha_sc,
                                         a_ref=PV_module_cec.a_ref,
                                         I_L_ref=PV_module_cec.I_L_ref,
                                         I_o_ref=PV_module_cec.I_o_ref,
                                         R_sh_ref=PV_module_cec.R_sh_ref,
                                         R_s=PV_module_cec.R_s,
                                         Adjust=PV_module_cec.Adjust)
```

In [ ]:

In [ ]:
```
#using the single diode equation to calculate the dc output parameteres
#current at maximum power point in amperes, voltage at maximum power point in volts
#power at maximum power point in watts

dc_result=pvlib.pvsystem.singlediode(*cec_params,method='newton')
dc_result[11:15]
```

PV system

In [ ]:

In [ ]:
```
#Now we will create our pvsystem in which there are 20 modules per string
#here we can change the modules per string and strings per inverter
```

```
system=PVSystem(modules_per_string=24,strings_per_inverter=1,surface_tilt=45,albed
```

In [ ]:
```
system
```

In [ ]:

In [ ]:
```
#as we know out dc_reuslt is for single module now we will scale our system
#this function Scales the voltage, current, and power of the DataFrame

dc_result_scaled=system.scale_voltage_current_power(dc_result)
p_mp_dc=dc_result_scaled['p_mp']
p_mp_dc[10:15]
```

In [ ]:
```
#plotting the dc output for a week

p_mp_dc_single_day=p_mp_dc.head(288)
p_mp_dc_single_day.plot(figsize=(10,6))
plt.xlabel('Days');
plt.ylabel('DC Power Output');
```

In [ ]:

In [ ]:
```
#Calculating the ac_power from PV watts model
#pdc0 from datasheet is 7300W inverter values from datasheet
#eta_inv_nom=0.97,
#eta_inv_ref=0.9637


ac_result=pvlib.inverter.pvwatts(dc_result_scaled.p_mp,
                                 pdc0=7300,
                                 eta_inv_nom=0.965,
                                 eta_inv_ref=0.9637)

ac_result.head(5)
```

In [ ]:
```
#plotting the trend for a week

single_day=ac_result.head(288)
single_day.plot(figsize=(10,6))
plt.xlabel('Days');
plt.ylabel('AC Power Output')
```

In [ ]:
```
#Comparing ac and dc output for single day

plt.figure(figsize=(10,6))
plt.plot(ac_result.head(288))
plt.plot(p_mp_dc_single_day.head(288))
plt.xlabel('Time');
plt.ylabel('Power Output')
```

In [ ]:
```
#We got the hourly ac power output for a PV system with 20 modules connected to an

ac_result.head(288)
```