IAM Model Validation

Lelia Deville - SAND2023-11359O

The purpose of this notebook is to validate the use of new or current IAM models through multiple steps of analysis

The model input data comes from the published results of the 2021 Blind Modeling Comparison. In the following notebook the data collected in S2 is used, which is from the Canadian Solar 275W system at Sandia National Labs in Albuquerque, NM from Jan 2021 - Dec 2021. More information about the dataset can be found at the DuraMAT Data Hub and the published results of the 2021 Blind Modeling Comparison

To demonstrate the way this notebook should work, a pvlib-python model, specifically the Physical IAM model, is used in place of a user defined model. To use this notebook for a custom model, simply replace the Physical model defined in *3: Run model to be validated or import model results*. If the model is run by an external program, it is also possible to import only the results to use in the validation and analysis.

The notebook is segemented into 4 sections:

- **1. Import data from DuraMAT**
- **2. Define relevant system & meteo data**
- **3. Run model to be validated or import model results**
- **4. Compare model to measured results and other baseline models**

In [1]:
```
#import necessary packages and set default formatting for plots
import matplotlib.pyplot as plt #v. 3.7.2
from matplotlib.lines import Line2D #v. 3.7.2
import numpy as np #v. 1.24.3
import seaborn as sns #v 0.12.2
import pandas as pd #v. 2.0.3
import pvlib #v. 0.9.3
from pvlib.tools import sind, cosd, acosd
import scipy #v. 1.11.1
from tabulate import tabulate #v. 0.8.10

plt.rcParams["figure.figsize"] = (10,6)
plt.rcParams['font.size']=12
plt.rcParams['lines.linewidth']=1.25
plt.rcParams['xtick.labelsize']=12
plt.rcParams['ytick.labelsize']=12
plt.rcParams['axes.titlesize']=12
pd.options.mode.chained_assignment = None
```

1. Import data

This section imports the meteo and system data from the DuraMAT Datahub. For the purpose of accurate solar position calculations, the times are set to be labeled at the middle-of-hour. The data includes 2 filters: *bsrn pass* and *SNL No Snow*. The baseline surface radiation network (BSRN) filter follows version 2 quality control tests and the SNL No Snow filter removes any days with recorded snow fall or snow depth. Data is removed if either filter value is '0'. For the meteo data, any 0 values are replaced with NaNs so that statistical values, like mean, are not affected by these values.

In [2]:
```
# read in data from duramat data hub directly
df = pd.read_excel("https://datahub.duramat.org/dataset/293db0cb-e838-4f7a-8e77-f62e85328c47/resource/b54bdc36-1864-48a9-abab-daf0e3f8dcf5/download/  \
            pvpmc_2021_blind_modeling_comparison_data_s1-s6.xlsx,sheet_name='S2'")
#Reassigning the index so the timesteps are at the middle of the hour
df.index = pd.date_range(start='2021-01-01 00:30:00', end='2021-12-31 23:30:00', freq='H')
df.index = df.index.tz_localize('MST')

#apply the filters that are included in the data & replacing any 0 with nan so they dont affect error metrics
#dropping nans helps keep size down so operations run more quickly and smoothly
df = df.where((df['bsrn_pass'] == 1) & (df['SNL No Snow'] == 1)).dropna()
df.replace(0, np.nan, inplace=True)
df.dropna(inplace=True)

df.head()
```

Out[2]:

	Scenario	Year	Month	Day	Hour	GHI (W/m2)	DNI (W/m2)	DHI (W/m2)	Ambient Temp (°C)	Relative Humidity (%)	Wind Speed (m/s)	Measured front POA irradiance (W/m2)	Measured module temperature (°C)	Measured DC power (W)	bsrn_pass	SNl No Snov
2021-01-01 08:30:00-07:00	S2	2020.0	1.0	1.0	9.0	185.738601	754.498236	31.546335	-3.652383	54.784333	1.803700	442.132104	6.645174	1292.814741	1.0	1.(
2021-01-01 09:30:00-07:00	S2	2020.0	1.0	1.0	10.0	353.666975	914.471581	40.138926	-0.708700	41.447333	2.923567	701.031595	17.712519	2276.603041	1.0	1.(
2021-01-01 10:30:00-07:00	S2	2020.0	1.0	1.0	11.0	482.624408	978.551782	44.586906	0.819633	38.089500	2.962067	879.164182	25.669461	2782.780150	1.0	1.(
2021-01-01 11:30:00-07:00	S2	2020.0	1.0	1.0	12.0	555.822941	1006.709614	44.024464	2.140700	36.223167	1.919817	977.788429	35.226433	2989.486270	1.0	1.(
2021-01-01 12:30:00-07:00	S2	2020.0	1.0	1.0	13.0	546.147743	865.317214	98.340036	3.236667	35.082167	1.641850	922.354253	38.056121	2796.495393	1.0	1.(

2. Define system and meteo data

'module' is a dictionary of module specific values for 275 W mono-Si Canadian Solar modules and includes system and module data. All data for this system can be found in the various reports on the PVPMC Wesbite. Solar position calculations generate azimuth, zenith, elevation, etc for every timestep in the df

In [3]:
```
#Defining constants and values that are consistent across all calculations
#we are using S2 from the data, which is the Candian Solar Monocrystalline 275W module
module = {'Tilt': 35,'Latitude': 35.05,'Longitude': -106.54,'Altitude': 1600,'Surface Azimuth': 180,'String Length':12, 'iam0':1,'iam10': 0.9989, 'iam20': 1.0014,
          'iam30': 1.0002, 'iam40':0.9984, 'iam45': 0.9941, 'iam50': 0.9911, 'iam55': 0.9815, 'iam60':0.9631, 'iam65':0.9352, 'iam70':0.8922, 'iam75':0.8134,
          'iam80':0.6778, 'iam85': 0.4351,'U0': 28.825, 'U1': 4.452, 'NOCT': 45, 'Unit Mass': 11.119,'Area':1.621,'Vmp': 31.48 ,'Imp': 8.81,'Voc':38.29 ,
          'Isc': 9.30,'Pmp': 275,'Gamma Pmp': -0.0041,'Alpha Isc':0.0033,'Beta Voc': -0.1178,'Cell Type':'monoSi','Cells in Series':60}
module = pd.Series(module)

#Running solar position calculations
spdf = pvlib.solarposition.get_solarposition(time=df.index, latitude=module['Latitude'],
       longitude=module['Longitude'],temperature=df['Ambient Temp (°C) '], altitude=module['Altitude'])
df['dni_extra'] = pvlib.irradiance.get_extra_radiation(datetime_or_doy=df.index)
pres = pvlib.atmosphere.alt2pres(module['Altitude'])
```

```
#Save these values into the df with inputs & results for use in later analysis
df['Azimuth'] = spdf['azimuth']
df['Zenith'] = spdf['apparent_zenith']
df['Sol Elev'] = spdf['elevation']
df['AOI'] = pvlib.irradiance.aoi(surface_tilt=module['Tilt'], surface_azimuth=module['Surface Azimuth'],
                    solar_zenith=spdf['apparent_zenith'], solar_azimuth=spdf['azimuth'])
df['Airmass'] = pvlib.atmosphere.get_relative_airmass(zenith=spdf['apparent_zenith'])
df['Clearness Index'] = pvlib.irradiance.clearness_index(ghi=df['GHI (W/m2)'], solar_zenith=spdf['apparent_zenith'], extra_radiation = df['dni_extra'])

#The 'true' values that the IAM models will be compared to are the interpolated IAMs found using module-specific measured data
ref_thetas= [0,10,20,30,40,45,50,55,60,65,70,75,80,85]
ref_iams = [module['iam0'],module['iam10'],module['iam20'],module['iam30'],module['iam40'],module['iam45'],module['iam50'],
            module['iam55'],module['iam60'],module['iam65'],module['iam70'],module['iam75'],module['iam80'],module['iam85']]
df['IAM - Interp'] = pvlib.iam.interp(aoi=df['AOI'],theta_ref=ref_thetas,iam_ref=ref_iams, method='cubic' )
spdf.head()
```

Out[3]:

	apparent_zenith	zenith	apparent_elevation	elevation	azimuth	equation_of_time
2021-01-01 08:30:00-07:00	77.884310	77.950122	12.115690	12.049878	129.546848	-3.734135
2021-01-01 09:30:00-07:00	69.241432	69.279260	20.758568	20.720740	140.756151	-3.753597
2021-01-01 10:30:00-07:00	62.615700	62.643406	27.384300	27.356594	154.026282	-3.773049
2021-01-01 11:30:00-07:00	58.731118	58.754688	31.268882	31.245312	169.230769	-3.792492
2021-01-01 12:30:00-07:00	58.153100	58.176057	31.846900	31.823943	185.427677	-3.811925

3. Run the model or import the results to be validated

A model can either be defined and run within this notebook or could be run externally and the results imported below. For demonstration purposes the `pvlib.iam.physical` function is used but should be replaced by the user's model.

In [4]:
```
#Either run a model in this notebook or import the results into the column name below

#run model here
df['Modeled IAM'] = pvlib.iam.physical(aoi=df['AOI'])

# or import model results here --- make sure timestamps line up and are middle-of-hour
# df['Modeled IAM'] = pd.read_excel('results.xlsx')

#specify a model name for use in analysis and plotting
model_name = 'Physical'
```

In [5]:
```
#checking for unphysical IAM & AOI values
print((df.loc[df['AOI'] < 20, 'Modeled IAM'] > 0.98).all())
print((df.loc[df['AOI'] > 90, 'Modeled IAM'] == 0).all())
print((df['Modeled IAM'] <= 1.0).all())
```

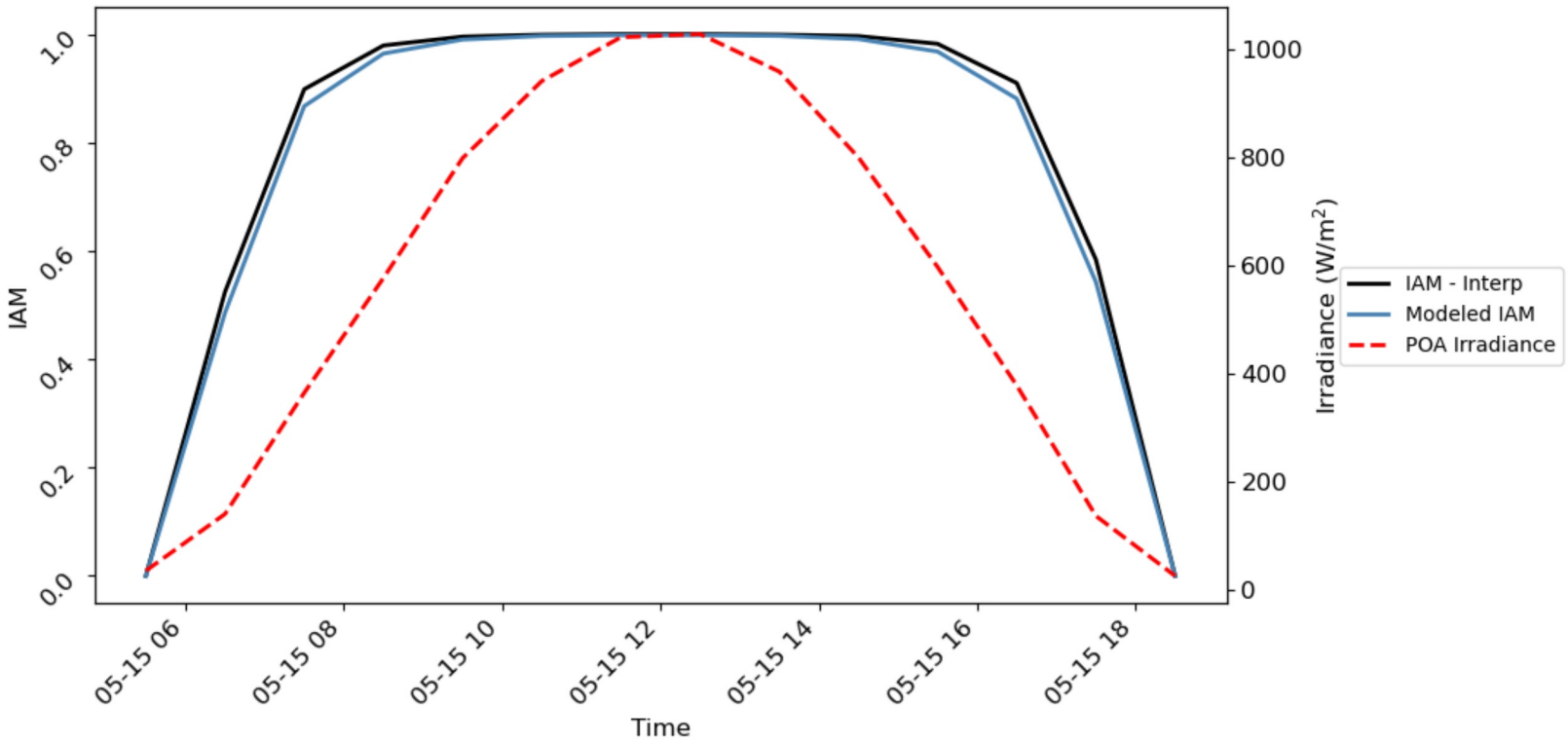
```
True
True
True
```

Visualize the results of the model over a sample day

This preliminary check helps make sure the results are feasible and there aren't any obvious errors like time shifts or magnitude differences

In [6]:
```
#diurnal plot
date = '2021-05-15'
fig, ax = plt.subplots()
ax2 = ax.twinx()
df.loc[date, 'IAM - Interp'].plot(ax=ax, label='IAM - Interp', linewidth=2, color='black', zorder=5.5)
df.loc[date, 'Modeled IAM'].plot(ax=ax, label='Modeled IAM', linewidth=2, color='steelblue', zorder=5.5)
df.loc[date, 'Measured front POA irradiance (W/m2)'].plot(ax=ax2, label='POA Irradiance', linewidth=2, color='red', zorder=2.5, linestyle='dashed')

line_1 = Line2D([0], [0], color='black', linewidth=2, linestyle='-',label='IAM - Interp')
line_3 = Line2D([0], [0], color='steelblue', linewidth=2, linestyle='-',label='Modeled IAM')
line_4 = Line2D([0], [0], color='red', linewidth=2, linestyle='--',label='POA Irradiance')
lines = [line_1,line_3,line_4]
plt.legend(prop=dict(size='small'), loc=[1.1, 0.4],handles=lines)
ax.set_ylabel('IAM')
ax.tick_params(labelrotation=45)
ax2.set_ylabel('Irradiance (W/m$^2$)')
ax.set_xlabel('Time')
```

Out[6]:
```
Text(0.5, 0, 'Time')
```



4. Compare modeled values to measured values + other baseline models

3 steps of analysis:

- **1. Overall MBE, RMSE, and other errors of the model**
- **2. Residual analysis**
- **3. Comparison to baseline model**

Analysis I: Overall errors of the model

• Mean Bias Error (MBE) - shows the estimation bias of the model

$$\frac{\sum_{i=1}^{N}(V_{modeled}-V_{measured})}{N_{observations}}$$

• Root Mean Squared Error (RMSE) - measures average difference between modeled and measured values

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(V_{modeled}-V_{measured})^2}$$

In [7]:
```
df['BE'] = (df['Modeled IAM'] - df['IAM - Interp'])
mbe = df['BE'].mean()
rmse = np.sqrt((((df.dropna()['IAM - Interp'] - df.dropna()['Modeled IAM'])**2).sum())/(len(df.dropna()['Modeled IAM'])))
d = [ ['MBE', str(round(mbe,3))],  ['RMSE',str(round(rmse,3))]]
print (tabulate(d, headers=["Metric", "Value"]))
```

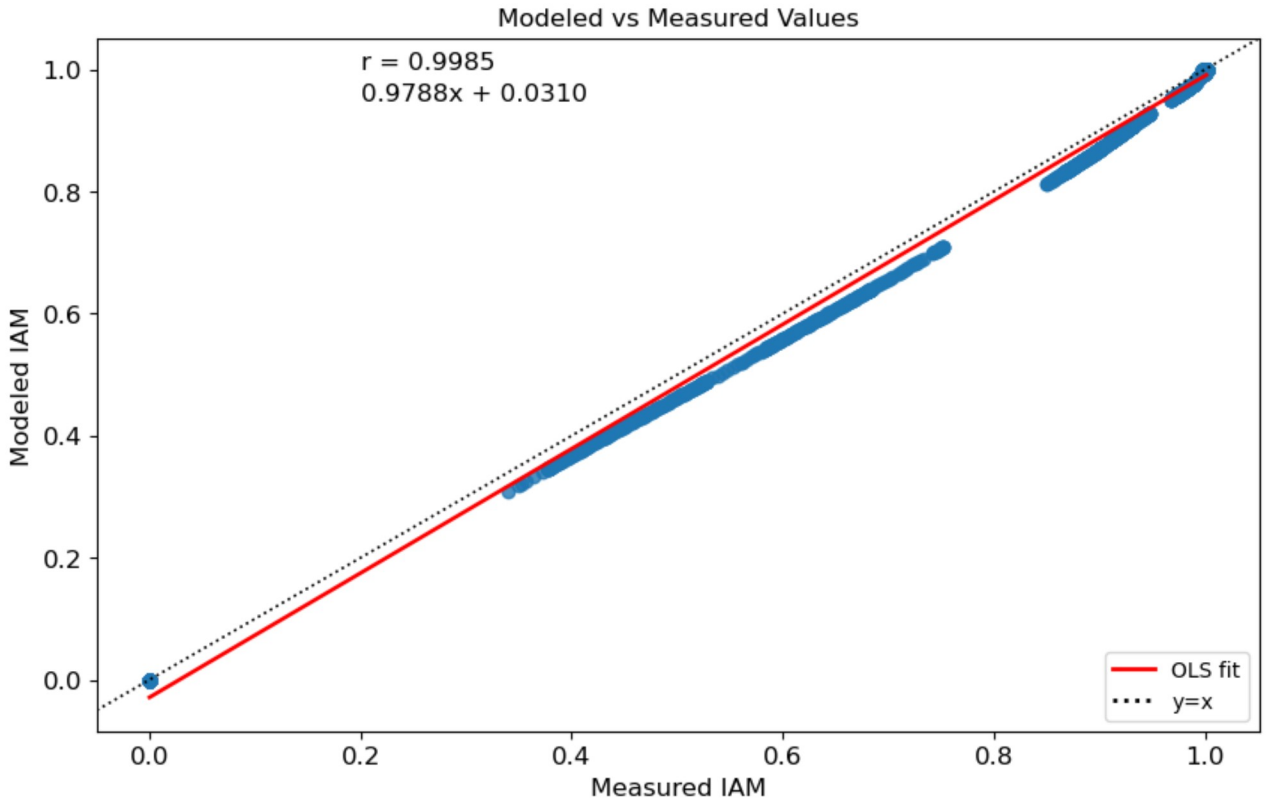
```
Metric      Value
--------  -------
MBE        -0.012
RMSE        0.018
```

Plotting the measured vs modeled values

The plot should be mostly linear. r and slope values close to one indicate good correlation and accurate model performance

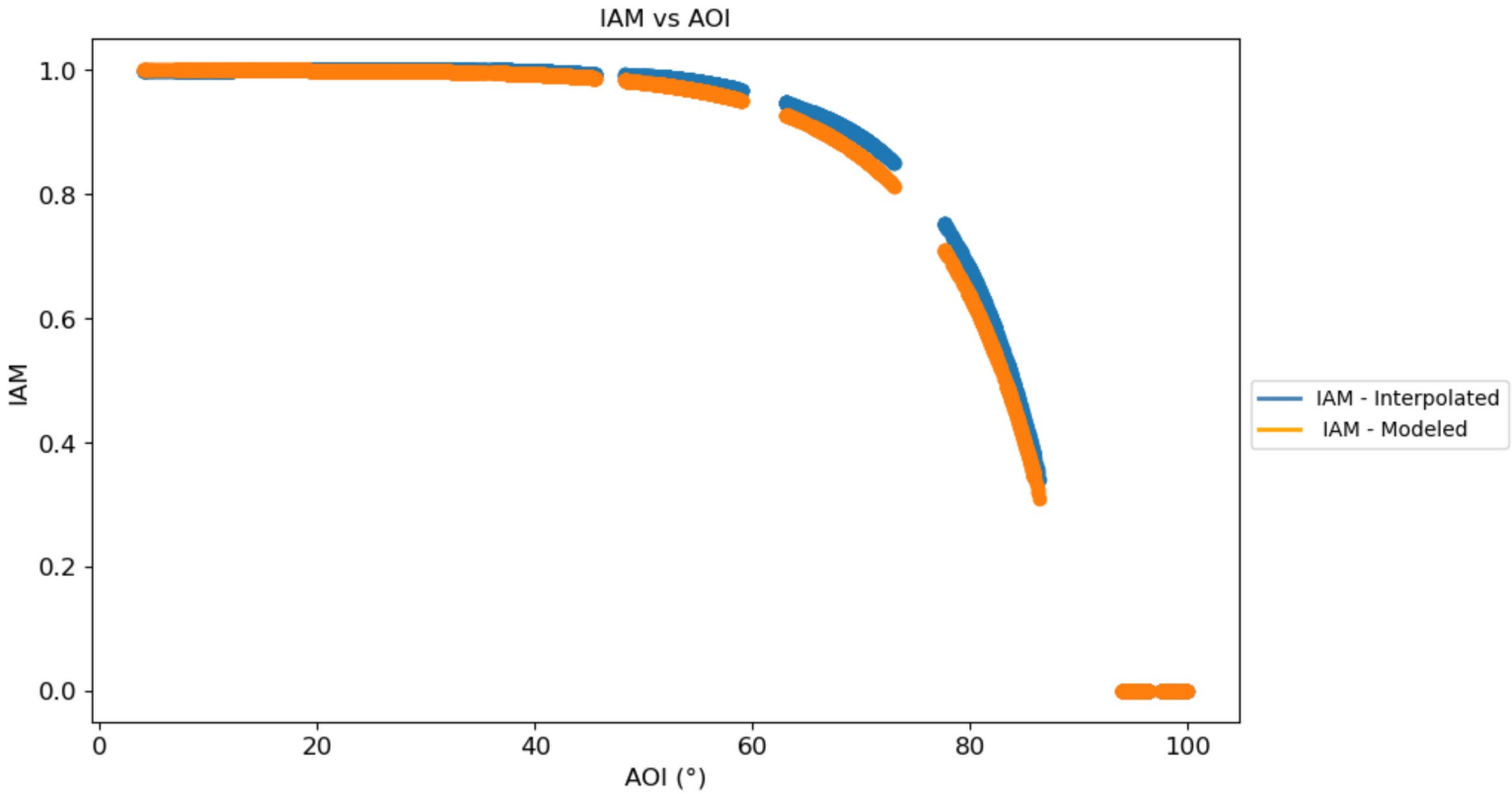
In [8]:
```
slope, intercept, r, p, std = scipy.stats.linregress(x = df.dropna()['Modeled IAM'], y = df.dropna()['IAM - Interp'])
sns.regplot(x = df['IAM - Interp'], y = df['Modeled IAM'], line_kws={'color':'red'})
plt.ylabel('Modeled IAM')
plt.xlabel('Measured IAM')
plt.text(0.2, 1.0, s = f"r = {r:0.04f}")
plt.text(0.2, 0.95, s = f"{slope:0.04f}x + {intercept:0.04f}")
plt.axline((0, 0), slope=1, c='k', ls=':')
line_1 = Line2D([0], [0], color='red', linewidth=2, linestyle='-',label='OLS fit')
line_2 = Line2D([0], [0], color='k', linewidth=2, linestyle=':',label='y=x')
plt.legend(prop=dict(size='small'), loc='lower right',handles=[line_1, line_2])
plt.title('Modeled vs Measured Values')
```

Out[8]: Text(0.5, 1.0, 'Modeled vs Measured Values')



In [9]:
```
#iam vs aoi curve
plt.scatter(x=df['AOI'], y=df['IAM - Interp'])
plt.scatter(x=df['AOI'], y=df['Modeled IAM'])
plt.ylabel('IAM')
plt.xlabel('AOI (°)')

line_1 = Line2D([0], [0], color='steelblue', linewidth=2, linestyle='-',label=('IAM - Interpolated'))
line_3 = Line2D([0], [0], color='orange', linewidth=2, linestyle='-',label= ' IAM - Modeled')
lines = [line_1,line_3]
plt.legend(prop=dict(size='small'), loc=[1.01, 0.4],handles=lines)
plt.title('IAM vs AOI')
```

Out[9]: Text(0.5, 1.0, 'IAM vs AOI')



Energy Yield Estimates

In order to see the impact of the IAM on overall energy output, we will run two performance models and used our measured (interpolated) values and the modeled IAM values and compare these outputs to see how they vary

To compare the IAM model, we will use the 'measured' IAM and modeled IAM as inputs into effective irradiance. To calculate this, the pvlib-python `pvlib.irradiance.get_total_irradiance` function with the Perez model is used to get direct and diffuse components of POA. The values are then used in the effective irradiance equation originally defined by King in 1983. In calcualting effecitive irradiance, spectral effects were neglected since there were no module-specific AM coefficients available.

In [10]:
```
#first we need poa components to use in calculating effective irradiance
efdf = pvlib.irradiance.get_total_irradiance(surface_tilt=module['Tilt'], surface_azimuth=module['Surface Azimuth'],
        solar_zenith=df['Zenith'], solar_azimuth=df['Azimuth'],dni=df['DNI (W/m2)'],
        ghi=df['GHI (W/m2)'], dhi=df['DHI (W/m2)'], dni_extra=df['dni_extra'], model ='perez', model_perez='albuquerque1988')
```

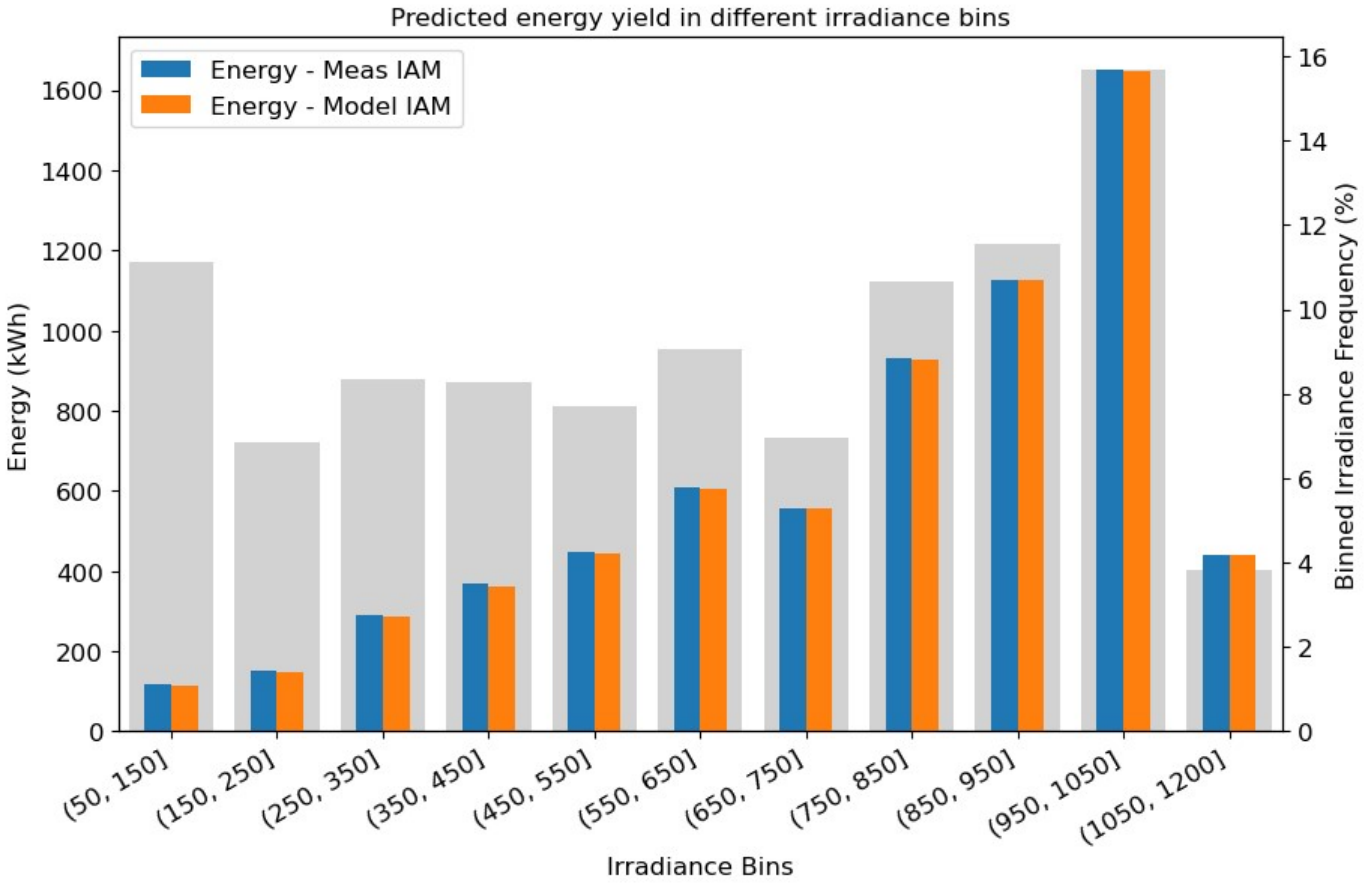
In [11]:
```
#use the interpolated & modeled IAM to get two different effective irradiance values, we will do further analysis on these in a bit

df['Effective Irradiance - Interp'] = ((efdf['poa_direct'] *  (df['IAM - Interp'])) + efdf['poa_diffuse'])
df['Effective Irradiance - Model IAM'] = ((efdf['poa_direct'] * (df['Modeled IAM'])) + efdf['poa_diffuse'])

df['Eff Irr - Model IAM NBE'] = 100 * (df['Effective Irradiance - Model IAM']-df['Effective Irradiance - Interp'])/(df['Effective Irradiance - Interp'])
```

In [12]:
```
#using effective irradiance with interpolated IAM to estimate energy
df['DC Power - Meas IAM'] = module['String Length']*pvlib.pvsystem.pvwatts_dc(g_poa_effective=df['Effective Irradiance - Interp'],
                                temp_cell=pvlib.temperature.sapm_cell_from_module(df['Measured module temperature (°C)'],
                            df['Measured front POA irradiance (W/m2)'], deltaT=3),pdc0=module['Pmp'], gamma_pdc=module['Gamma Pmp'])
ann_energy_meas = round(df['DC Power - Meas IAM'].sum()/1000,3)
#using effective irradiance with modeled IAM to estimate energy
df['DC Power - Model IAM'] = module['String Length']*pvlib.pvsystem.pvwatts_dc(g_poa_effective=df['Effective Irradiance - Model IAM'] ,
                                temp_cell=pvlib.temperature.sapm_cell_from_module(df['Measured module temperature (°C)'],
                            df['Measured front POA irradiance (W/m2)'], deltaT=3),pdc0=module['Pmp'], gamma_pdc=module['Gamma Pmp'])
ann_energy_model = round(df['DC Power - Model IAM'].sum()/1000,3)
#find overall % diff for annual energy
print('With interpolated IAM, predicted annual energy is',ann_energy_meas ,
      'kWh and with modeled IAM, predicted annual energy is', ann_energy_model, 'kWh')
print('The % difference in energy estimate when using interpolated vs modeled IAM is ',
      round(((ann_energy_model-ann_energy_meas)/ann_energy_meas)*100,3),'%')
```

```
With interpolated IAM, predicted annual energy is 6707.791 kWh and with modeled IAM, predicted annual energy is 6673.457 kWh
The % difference in energy estimate when using interpolated vs modeled IAM is  -0.512 %
```

In [13]:
```
#we can plot the energy produced in each bin of irradiance and see where the largest differences are when using modeled/measured IAM

df['Irradiance Bins']=(pd.cut(x=df['Measured front POA irradiance (W/m2)'], bins=[50,150,250,350,450,550,650,750,850,950,1050,1200]))
binstr = ['(50, 150]','(150, 250]','(250, 350]','(350, 450]','(450, 550]','(550, 650]','(650, 750]','(750, 850]','(850, 950]','(950, 1050]','(1050, 1200]']

bins = df['Irradiance Bins'].value_counts()
bins = bins.to_frame()
bins = bins.rename(columns = {'count':'Frequency'})
bins['Irradiance Bins'] = bins.index
bins.index.names = ['Index']
bins['Freq Norm'] =( bins['Frequency']/bins['Frequency'].sum()) * 100

bins['Energy - Model IAM'] = df.groupby('Irradiance Bins', observed=False).sum()['DC Power - Model IAM']/1000
bins['Energy - Meas IAM'] = df.groupby('Irradiance Bins', observed=False).sum()['DC Power - Meas IAM']/1000
bins = bins.sort_values('Irradiance Bins')

ax = bins.plot(x="Irradiance Bins", y=["Energy - Meas IAM", "Energy - Model IAM"], kind="bar", rot=0)

plt.xticks(rotation=30, ha='right')
ax.set_ylabel('Energy (kWh)')
ax.set_xlabel('Irradiance Bins')

ax2 = ax.twinx()
ax2 = sns.barplot(x='Irradiance Bins', y='Freq Norm', data=bins, errorbar=None, color='grey', alpha=0.35, zorder=2.5)
ax2.set_ylabel('Binned Irradiance Frequency (%)')
plt.grid(False)
plt.xticks(rotation=30, ha='right')
ax.set_zorder(ax2.get_zorder()+1)
ax.patch.set_visible(False)
```

```
plt.title('Predicted energy yield in different irradiance bins')
plt.show()
```



Analysis II: Residual Analysis

• Residual Analysis - quantifies the degree that variables may affect model errors

$$V_{modeled} - V_{measured}$$

Because of the sensitivity in taking differences in fractional values, we will be using the calcualted effective irradiances to analyze the residuals. All inputs were identical with the exception of the IAM, so any differences between the two can be attributed directly to IAM

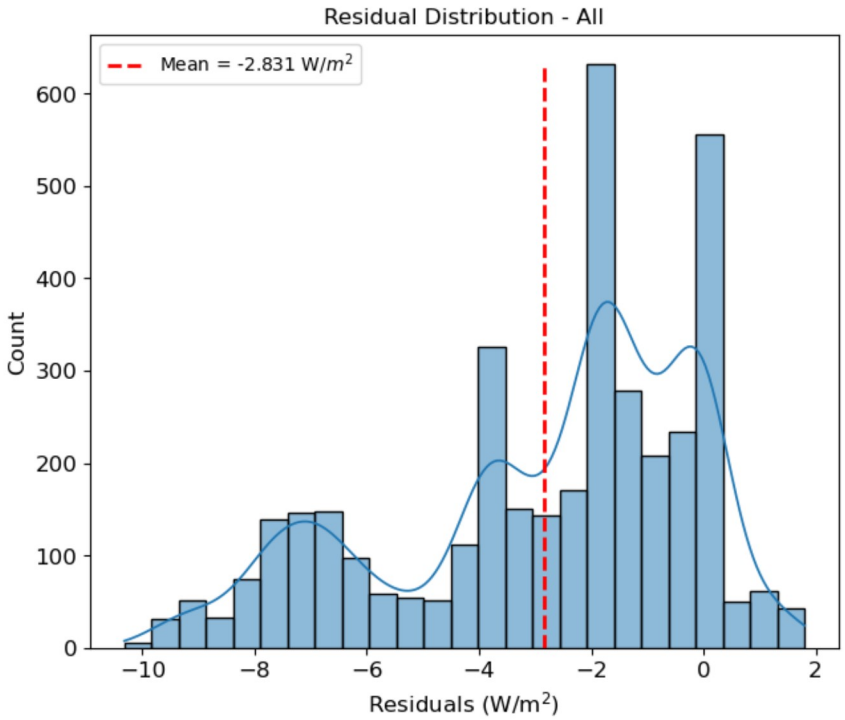
Residual Distribution

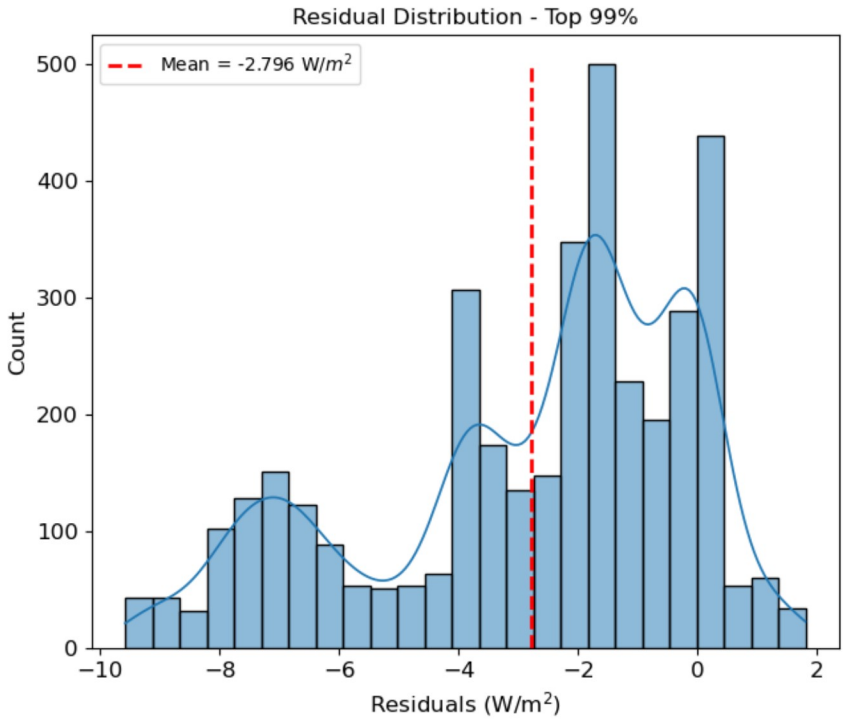
Residuals should be normally distributed, otherwise this indicates a consistent bias of over or under predicting

To get a closer look at a majority of the residuals, the outer 1% are removed using z-score. The distribution should be centered about the mean, shown by the red line

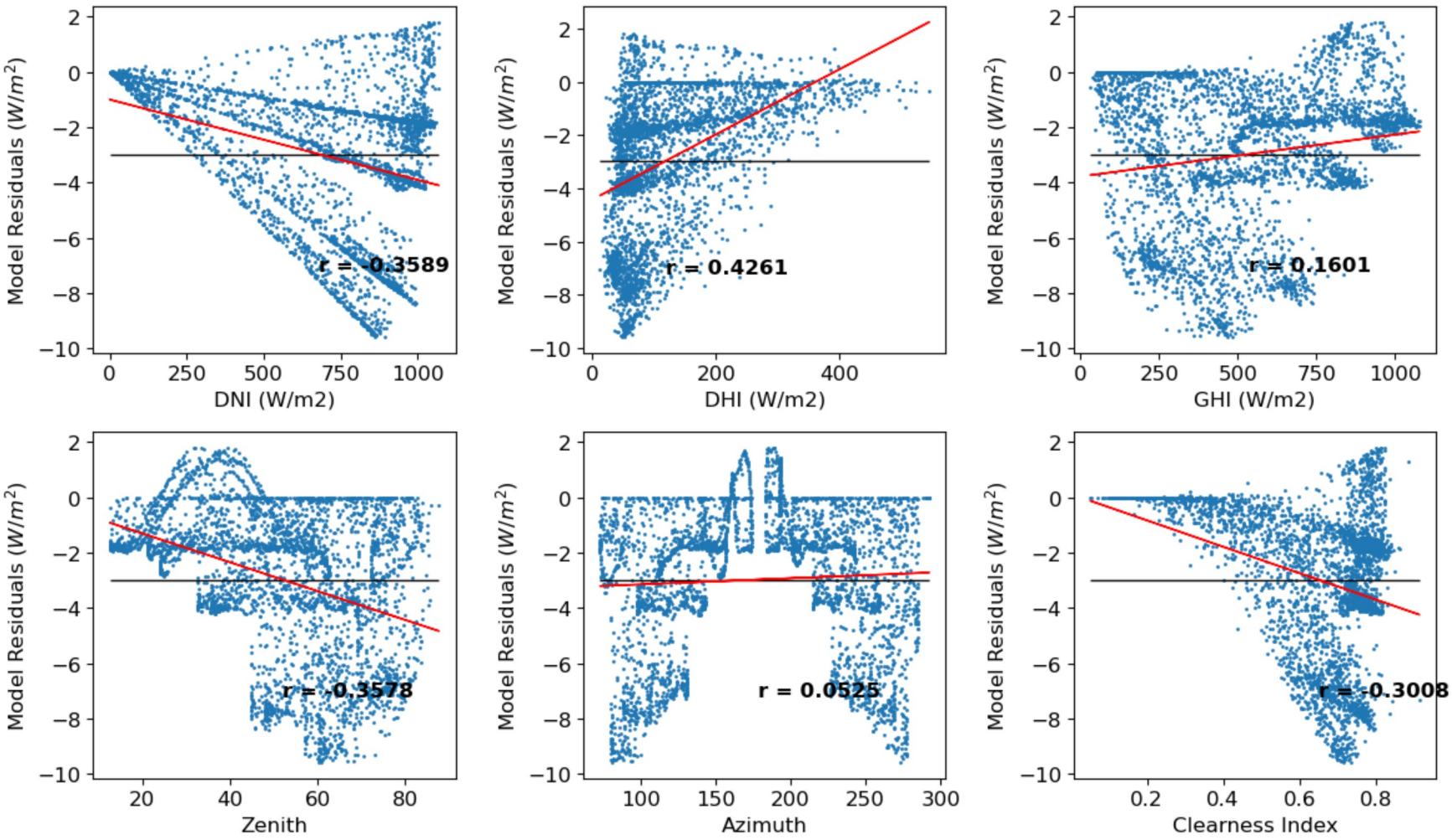
In [14]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,6))

df['Residuals'] = (df['Effective Irradiance - Model IAM'] - df['Effective Irradiance - Interp'])
hsp = sns.histplot(df['Residuals'], kde=True, bins=25, ax=ax1)
h = []
for rectangle in hsp.patches:
    h.append(rectangle.get_height())
ax1.vlines(x=df['Residuals'].mean(), ymin=0, ymax=max(h), linewidth=2, color='red',linestyles='--')
ax1.set_title('Residual Distribution - All')
ax1.set_xlabel('Residuals (W/m$^2$)')
line_4 = [Line2D([0], [0], color='red', linewidth=2, linestyle='--',label=('Mean ='+ '+str(round(df['Residuals'].mean(),3))+' W/$m^2$'))]
ax1.legend(prop=dict(size='small'),handles=line_4)

#Use z-score to eliminate the outer 1% of residuals
df['zscore'] = scipy.stats.zscore(df['Residuals'].dropna())
df['resid_trim'] = df['Residuals'][(df['zscore'] < 2.5) & (df['zscore'] > -2.5)]
hsp = sns.histplot(df['resid_trim'], kde=True, bins=25, ax=ax2)
h = []
for rectangle in hsp.patches:
    h.append(rectangle.get_height())
ax2.vlines(x=df['resid_trim'].mean(), ymin=0, ymax=max(h), linewidth=2, color='red',linestyles='--')
ax2.set_title('Residual Distribution - Top 99%')
ax2.set_xlabel('Residuals (W/m$^2$)')
line_4 = [Line2D([0], [0], color='red', linewidth=2, linestyle='--',label=('Mean ='+ '+str(round(df['resid_trim'].mean(),3))+' W/$m^2$'))]
ax2.legend(prop=dict(size='small'),handles=line_4)
```

Out[14]:
```
<matplotlib.legend.Legend at 0x18a4a70ebc0>
```





In [15]:
```
#plot residuals against common inputs into irradiance models - high correlation could indicate a weakness in the model's consideration of that variable
df = df.dropna()
covariates = ['DNI (W/m2)', 'DHI (W/m2)', 'GHI (W/m2)','Zenith', 'Azimuth', 'Clearness Index']
y = df['resid_trim']
y_avg = df['resid_trim'].mean()
y_med = df['resid_trim'].median()

fig, axes = plt.subplots(2, 3, figsize=(12, 7))
for covariate, ax in zip(covariates, axes.flatten()):
    x = df[covariate]
    z = np.polyfit(x, y, 1)
    p = np.poly1d(z)
    r = np.corrcoef(x, y)[0][1]

    ax.scatter(x, y, s=1)
    ax.hlines(y=y_avg, xmin=x.min(), xmax=x.max(), linewidth=1, color='black', linestyles='-')
    ax.text(x=x.mean(), y=(y.min() + (-0.25*y.min())), s=f"r = {r:0.04f}", weight='bold')
    ax.plot(x, p(x), linewidth=1, color='red')
    ax.set_xlabel(covariate)
    ax.set_ylabel('Model Residuals ($W/m^2$)')

fig.tight_layout()
```



Analemma Plots

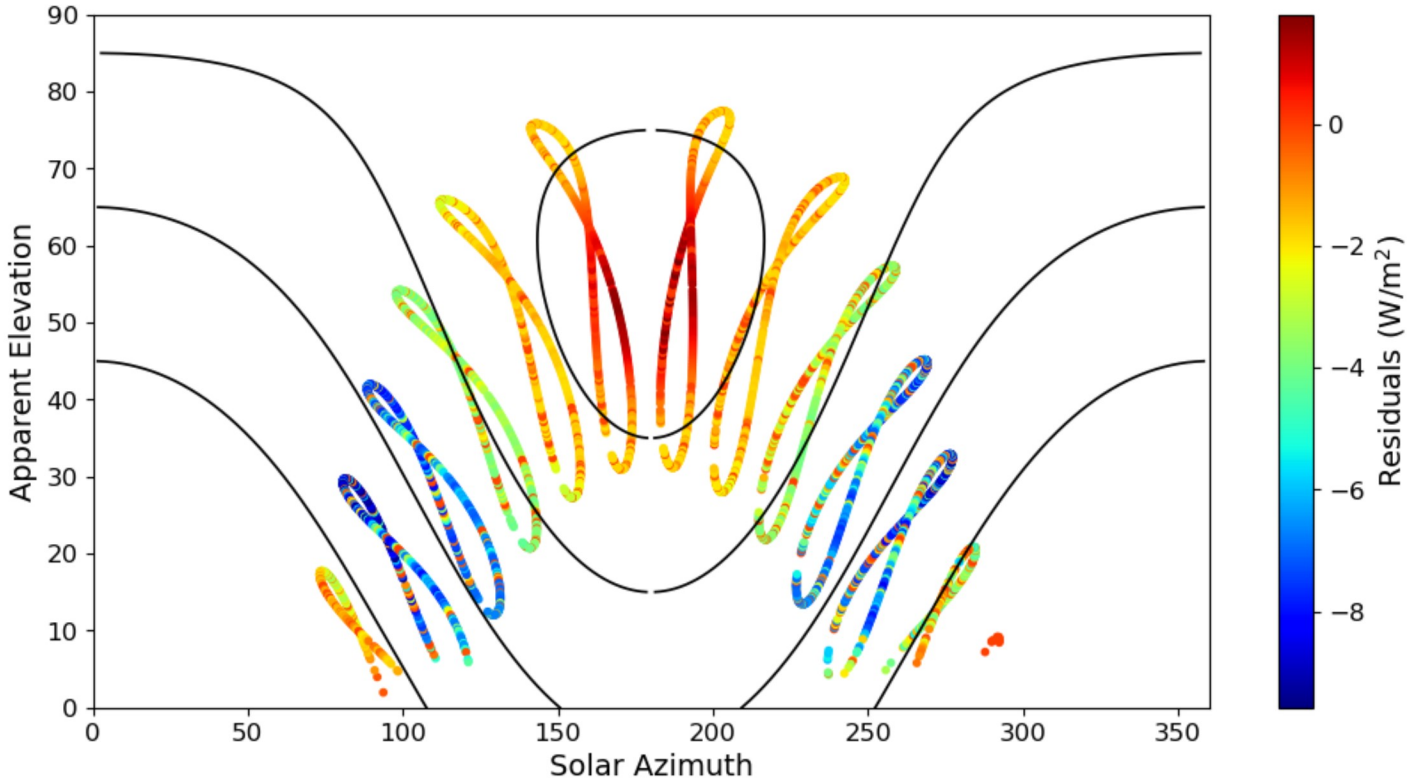
These are another way to check seasonality of a model and can also show how the model performs at specific times of day throughout the entire year

In [16]:
```
#analemma plots show the residuals at different times of the day/year with constant AOI contours
plt.figure(figsize=(12,6))
plt.scatter(x=df['Azimuth'], y=df['Sol Elev'], c=df['resid_trim'], cmap='jet', s=10)
for target_aoi in [20, 40, 60, 80]:
    zenith = np.linspace(0, 90, 5000)

    arccos_term = acosd(
        (cosd(target_aoi) - cosd(module['Tilt']) * cosd(zenith)) /
        (sind(module['Tilt']) * sind(zenith))
    )
```

```
    plt.plot((module['Surface Azimuth'] + arccos_term) % 360, 90 - zenith, c='k')
    plt.plot((module['Surface Azimuth'] - arccos_term) % 360, 90 - zenith, c='k')
clb = plt.colorbar()
clb.ax.set_ylabel('Residuals (W/m$^2$)',fontsize =14)
# plt.clim(0,(df['resid_trim']..quantile(0.75) + df['resid_trim'].std()))
plt.xlim(0,360)
plt.ylim(0,90)
plt.ylabel('Apparent Elevation', fontsize=14 )
plt.xlabel('Solar Azimuth',fontsize =14)
```

```
C:\Users\lmdevil\AppData\Local\Temp\ipykernel_11028\2502447864.py:8: RuntimeWarning: divide by zero encountered in divide
  (cosd(target_aoi) - cosd(module['Tilt']) * cosd(zenith)) /
C:\Users\lmdevil\Anaconda3\envs\pvmodel\lib\site-packages\pvlib\tools.py:103: RuntimeWarning: invalid value encountered in arccos
  res = np.degrees(np.arccos(number))
```

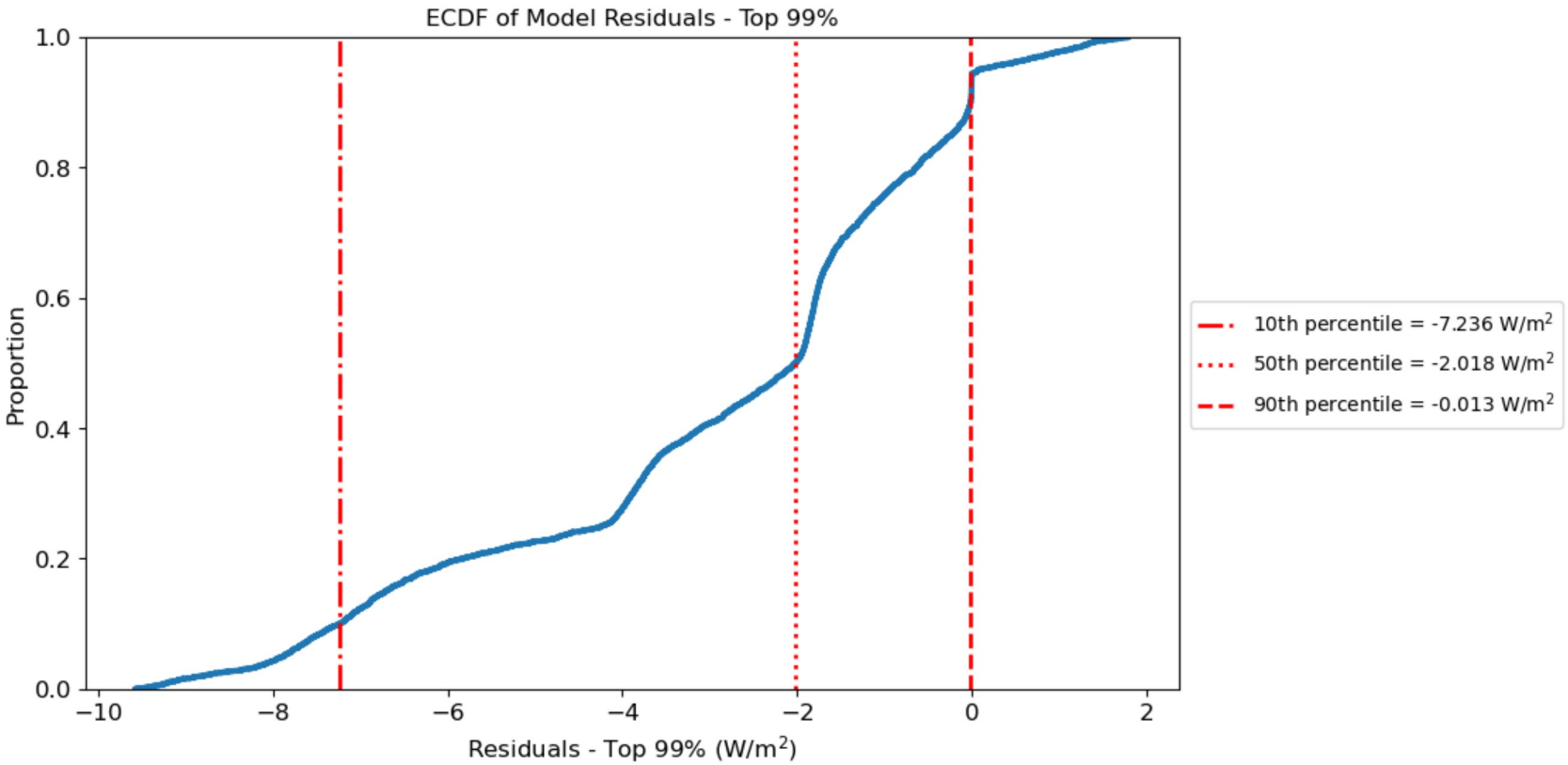
Out[16]: Text(0.5, 0, 'Solar Azimuth')



In [17]:
```
#plot empirical cumulative distribution functions - another way to visualize the distribution of the residuals
sns.ecdfplot(data=df, x='resid_trim', linewidth=3)
plt.xlabel('Residuals - Top 99% (W/m$^2$)')
perc10 = df['resid_trim'].quantile(0.1)
perc50 = df['resid_trim'].quantile(0.5)
perc90 = df['resid_trim'].quantile(0.9)
plt.axvline(perc10, linewidth=2, color='red', linestyle='-.', label=f'10th percentile = {perc10:0.03f} W/m$^2$')
```

```
plt.axvline(perc50, linewidth=2, color='red', linestyle='dotted', label=f'50th percentile = {perc50:0.03f} W/m$^2$')
plt.axvline(perc90, linewidth=2, color='red', linestyle='--', label=f'90th percentile = {perc90:0.03f} W/m$^2$')
plt.legend(prop=dict(size='small'), loc=[1.01, 0.4])
plt.title('ECDF of Model Residuals - Top 99%')
```

Out[17]: Text(0.5, 1.0, 'ECDF of Model Residuals - Top 99%')

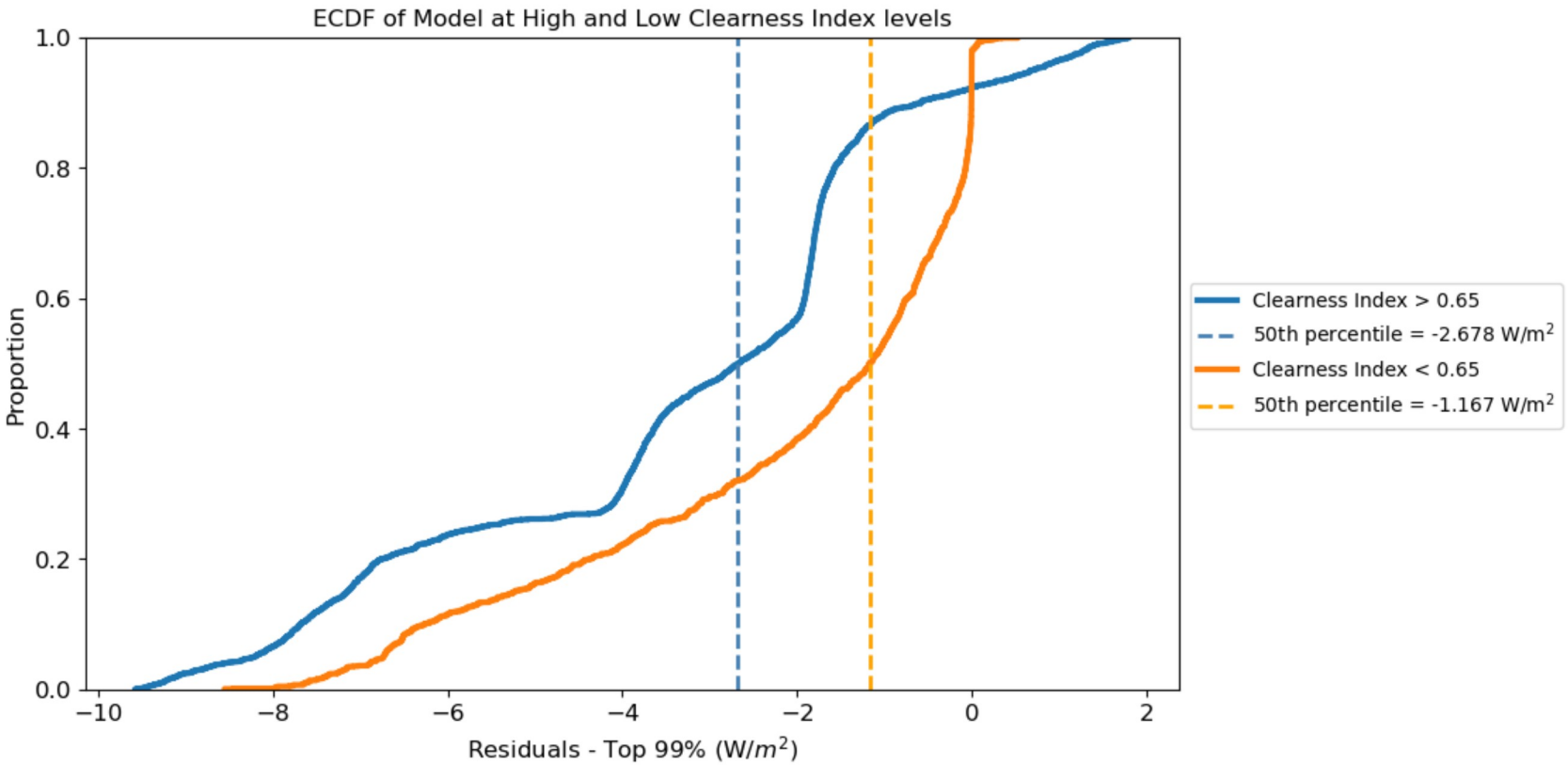


Plotting ECDF of model residuals with the division of some metric

In [18]:
```
metric = 'Clearness Index' #<---------------- could be any value that is a column in the df (wind speed, clearness index, ambient temp)
bound = 0.65 #<------------------------------ the bound at which to separate the upper and lower categories

df_h = df[df[metric] > bound]
df_l = df[df[metric] < bound]

perc50_h = df_h['resid_trim'].quantile(0.5)
perc50_l = df_l['resid_trim'].quantile(0.5)

sns.ecdfplot(data=df_h, x='resid_trim', linewidth=3, label=(metric+' > '+str(bound)))
plt.axvline(perc50_h, linewidth=2, color='steelblue', linestyle='--', label=f'50th percentile = {perc50_h:0.03f} W/m$^2$')
sns.ecdfplot(data=df_l, x='resid_trim', linewidth=3, label = (metric+' < '+str(bound)))
plt.axvline(perc50_l, linewidth=2, color='orange', linestyle='--', label=f'50th percentile = {perc50_l:0.03f} W/m$^2$')
```

```
plt.legend(prop=dict(size='small'), loc=[1.01, 0.4])
plt.xlabel('Residuals - Top 99% (W/$m^2$)')
plt.title('ECDF of Model at High and Low '+metric+' levels')
```

Out[18]: Text(0.5, 1.0, 'ECDF of Model at High and Low Clearness Index levels')



Analysis III: Comparison to Baseline Models

Comparing the model to other well-known baseline models can provide information about how the model is performing relative to accepted models. The baseline IAM model selected was the `pvlib.iam.ashrae` model

In [19]:
```
baseline_model = 'ASHRAE'
df['Baseline Model IAM'] = pvlib.iam.ashrae(aoi = df['AOI'])
#calcualte effective irradiance with the baseline iams
df['Effective Irradiance - Baseline'] = ((efdf['poa_direct'] *  (df['Baseline Model IAM'])) + efdf['poa_diffuse'])
#calculate some basic error metrics using that baseline effective irradiance
df['Baseline Residuals'] = df['Effective Irradiance - Baseline'] - df['Effective Irradiance - Interp']
df['Eff Irr - Baseline NBE'] = 100 * (df['Effective Irradiance - Baseline']-df['Effective Irradiance - Interp'])/df['Effective Irradiance - Interp']
```

In [20]:
```
#using modeled POA to estimate energy
df['DC Power - Baseline Model IAM'] = module['String Length']*pvlib.pvsystem.pvwatts_dc(g_poa_effective=df['Effective Irradiance - Baseline'],
```

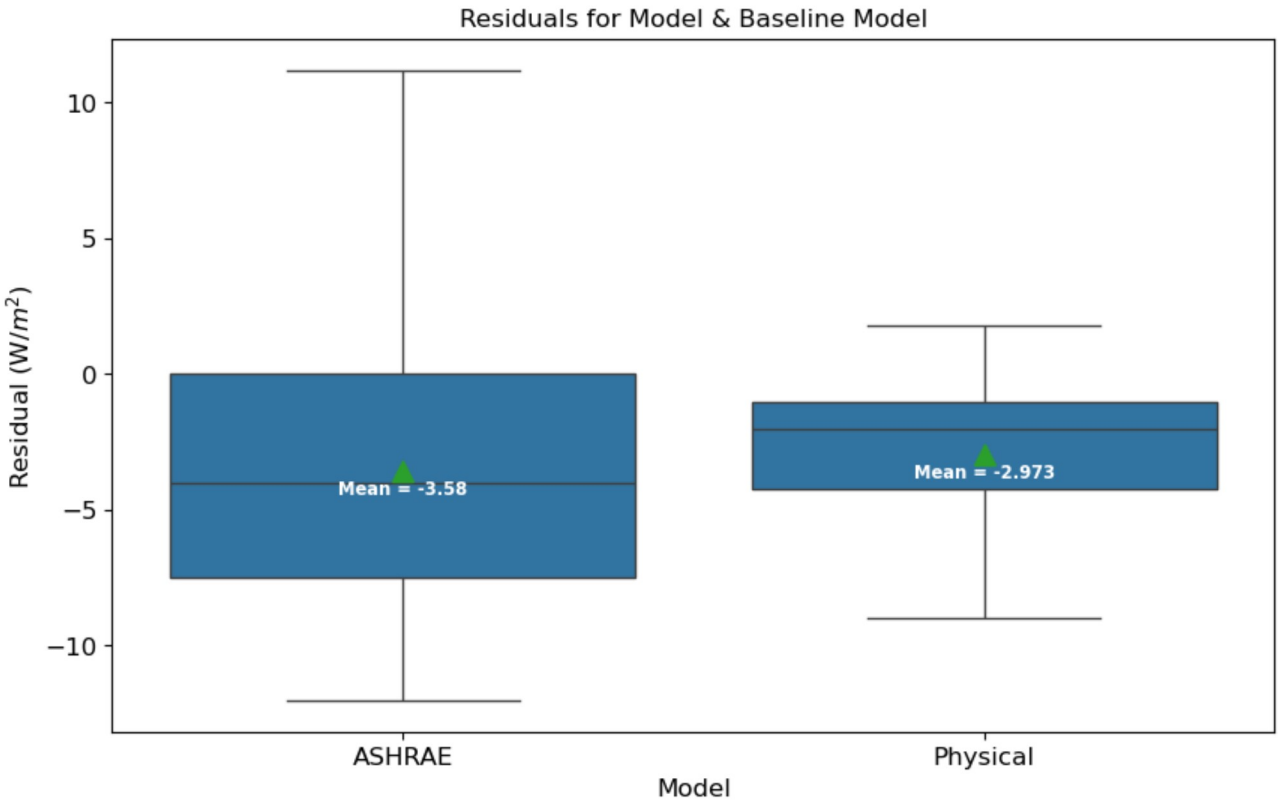
```
                                    temp_cell=pvlib.temperature.sapm_cell_from_module(df['Measured module temperature (°C)'], df['Modeled IAM'], deltaT=3),
                                    pdc0=module['Pmp'], gamma_pdc=module['Gamma Pmp'])
ann_energy_baseline = round(df['DC Power - Baseline Model IAM'].sum()/1000,3)
#find overall % diff for annual energy
print('With initial model IAM, predicted annual energy is', ann_energy_model,
      'kWh and with baseline modeled IAM, predicted annual energy is',ann_energy_baseline , 'kWh')
print('The % difference in energy estimate when using baseline vs modeled IAM is ', round(((ann_energy_baseline-ann_energy_model)/ann_energy_model)*100,3),'%')
```

```
With initial model IAM, predicted annual energy is 6673.457 kWh and with baseline modeled IAM, predicted annual energy is 6699.868 kWh
The % difference in energy estimate when using baseline vs modeled IAM is  0.396 %
```

In [21]:

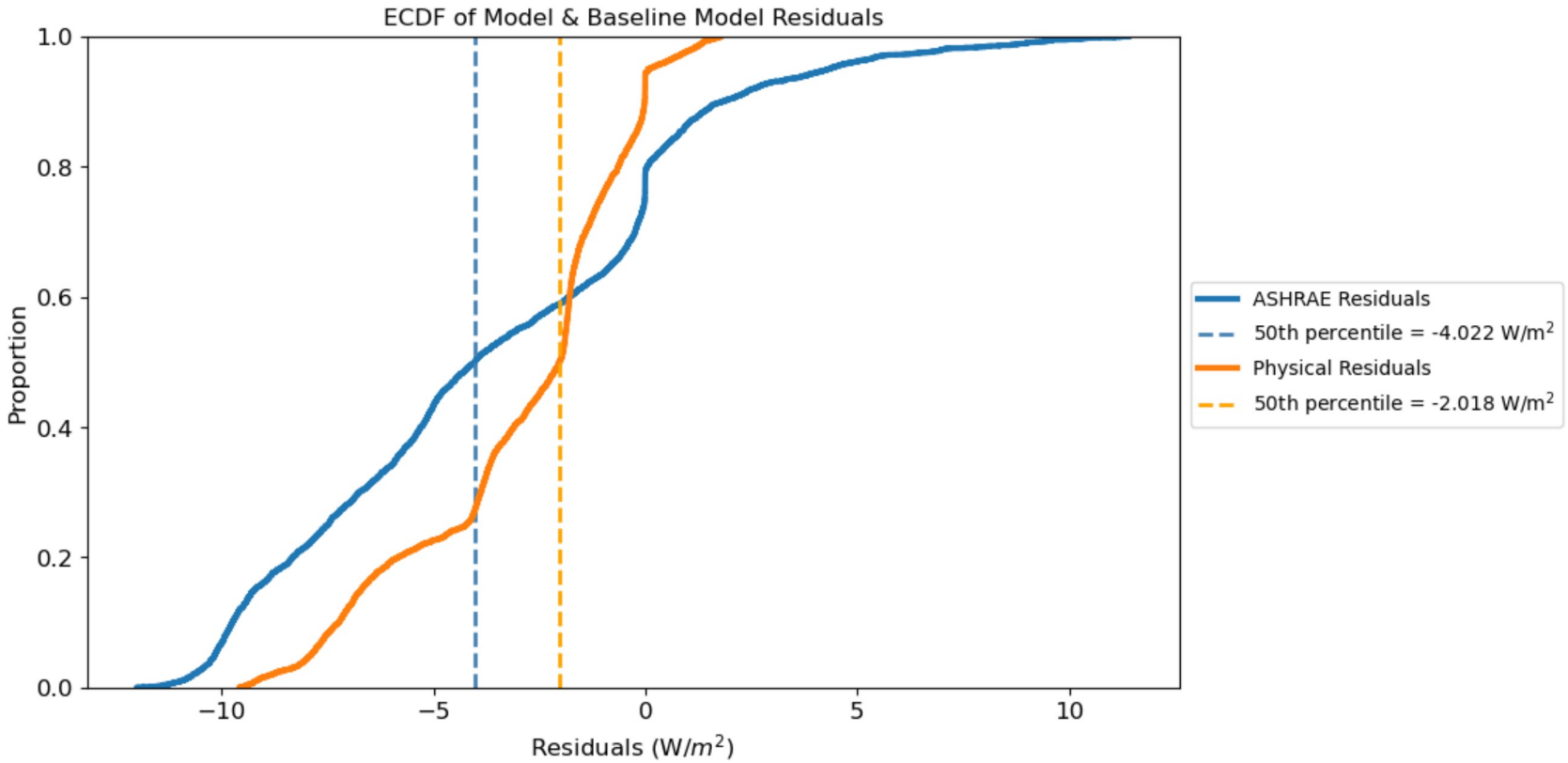
```
#put the model and baseline model residuals in one df for easy analysis
resid_df = pd.concat([
    pd.DataFrame({'Residual': df['Effective Irradiance - Baseline'] - df['Effective Irradiance - Interp'], 'Model': baseline_model,}),
    pd.DataFrame({'Residual': df['Effective Irradiance - Model IAM'] - df['Effective Irradiance - Interp'],'Model':model_name ,})
], ignore_index=True)

box_plot = sns.boxplot(x='Model', y='Residual', data=resid_df, showfliers=False, showmeans=True, meanprops={'markersize':10})
plt.ylabel('Residual (W/$m^2$)')
#view the numerical value of median on plot
means = resid_df.groupby(['Model'])['Residual'].mean()
vertical_offset = resid_df['Residual'].mean() * 0.25 # offset from median for display
for xtick in box_plot.get_xticks():
    if xtick == 0:
        name = baseline_model
    else:
        name = model_name
    box_plot.text(xtick,means[name] + vertical_offset,('Mean = '+str(round(means[name],3))),horizontalalignment='center',size='x-small',color='w',weight='semibol
plt.title('Residuals for Model & Baseline Model')
```

Out[21]: Text(0.5, 1.0, 'Residuals for Model & Baseline Model')

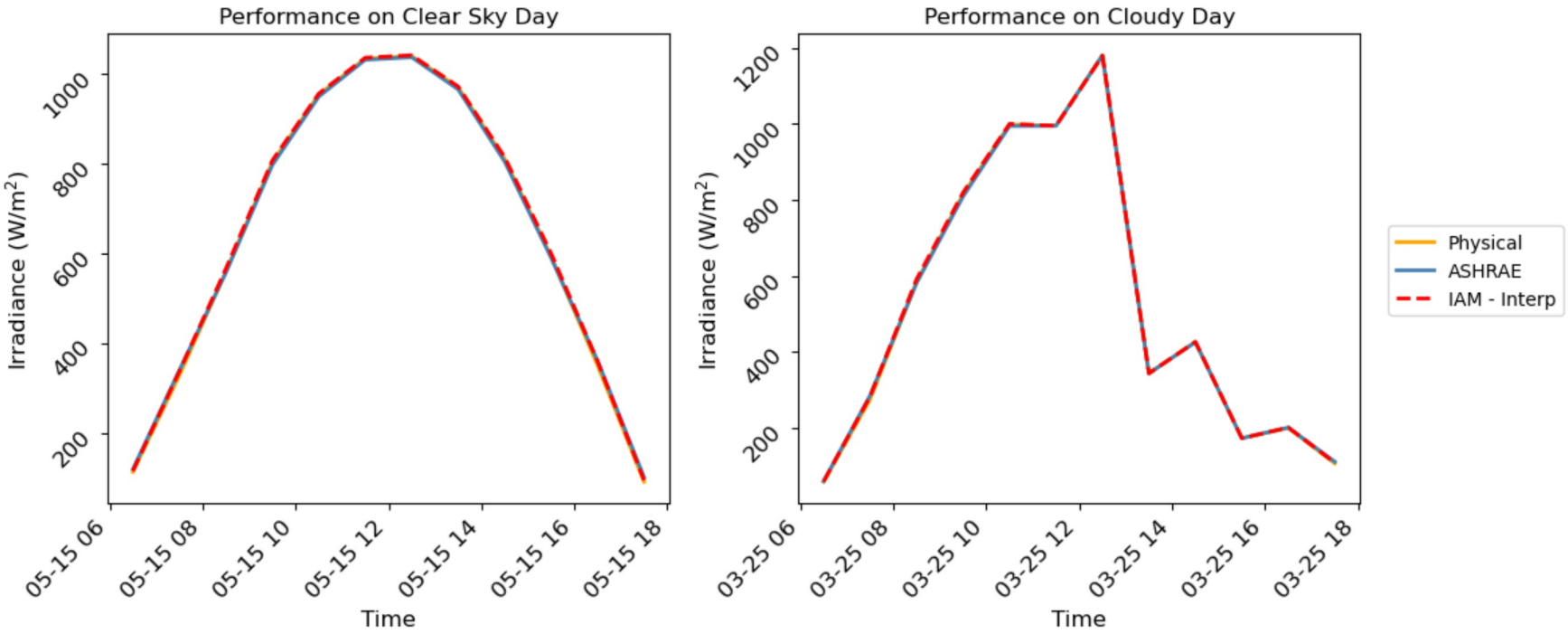


In [22]:
```
#ecdf of the two models overlayed & p50 for each

perc50_m = np.percentile(df['Residuals'].dropna(), 50)
perc50_b = np.percentile(df['Baseline Residuals'].dropna(), 50)

sns.ecdfplot(data=df, x='Baseline Residuals', linewidth=3, label = (baseline_model+' Residuals'))
plt.axvline(x=perc50_b, linewidth=2, color='steelblue', linestyle='--', label=f'50th percentile = {perc50_b:0.03f} W/m$^2$')
sns.ecdfplot(data=df, x='Residuals', linewidth=3, label=(model_name+' Residuals'))
plt.axvline(x=perc50_m, linewidth=2, color='orange', linestyle='--', label=f'50th percentile = {perc50_m:0.03f} W/m$^2$')
plt.legend(prop=dict(size='small'), loc=[1.01, 0.4])
plt.xlabel('Residuals (W/$m^2$)')
plt.title('ECDF of Model & Baseline Model Residuals')
```

Out[22]:
```
Text(0.5, 1.0, 'ECDF of Model & Baseline Model Residuals')
```



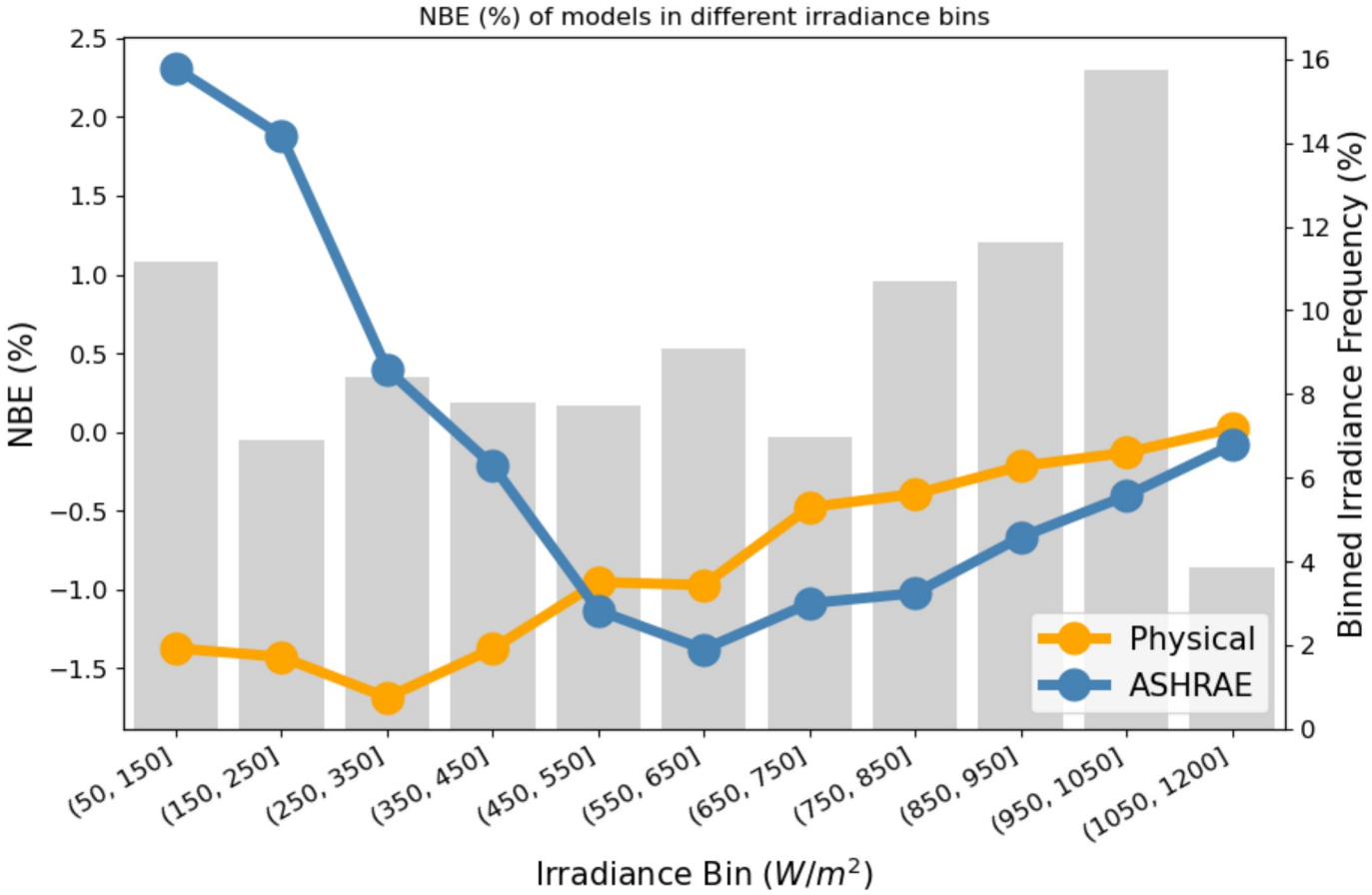
In [23]:
```
# diurnal plots help visualize the differences between effective irradiance using modeled and measured IAM as well as
# effective irradiance using model and baseline model IAM performance

dates = [('Clear Sky', '2021-05-15'),('Cloudy', '2021-03-25'),]

fig, axes = plt.subplots(1, len(dates), figsize=(12,5))

for (sky_condition, date), ax in zip(dates, axes):
    df.loc[date, 'Effective Irradiance - Model IAM'].plot(ax=ax, linewidth=2, color='orange', label = model_name)
    df.loc[date, 'Effective Irradiance - Baseline'].plot(ax=ax, linewidth=2, color='steelblue',label = baseline_model)
    df.loc[date, 'Effective Irradiance - Interp'].plot(ax=ax, linewidth=2, linestyle='dashed', color='red', label = 'IAM - Interp')
    ax.tick_params(labelrotation = 45)
    ax.set_ylabel('Irradiance (W/m$^2$)')
    ax.set_xlabel('Time')
    ax.set_title(f'Performance on {sky_condition} Day')

axes[-1].legend(prop=dict(size='small'), loc=[1.05, 0.4])
fig.tight_layout()
```



In [24]:
```
#view the model and baseline model performance at different levels of irradiance

df['Irradiance Bins']=(pd.cut(x=df['Measured front POA irradiance (W/m2)'], bins=[50,150,250,350,450,550,650,750,850,950,1050,1200]))
binstr = ['(50, 150]','(150, 250]','(250, 350]','(350, 450]','(450, 550]','(550, 650]','(650, 750]','(750, 850]','(850, 950]','(950, 1050]','(1050, 1200]']

bins = df['Irradiance Bins'].value_counts().to_frame()
bins = bins.rename(columns = {'count':'Frequency'})
bins['Irradiance Bins'] = bins.index
bins.index.names = ['Index']
bins['Freq Norm'] =( bins['Frequency']/bins['Frequency'].sum()) * 100
bins['Freq Norm'].sum()

fig, ax = plt.subplots()
x = binstr
y = df[['Irradiance Bins','Eff Irr - Model IAM NBE']].groupby('Irradiance Bins', observed=False).mean().sort_values('Irradiance Bins')['Eff Irr - Model IAM NBE']
ax.plot(x, y, 'orange', marker='o', zorder=6.5, linewidth=5, markersize=15)
y = df[['Irradiance Bins','Eff Irr - Baseline NBE']].groupby('Irradiance Bins', observed=False).mean().sort_values('Irradiance Bins')['Eff Irr - Baseline NBE']
ax.plot(x, y, 'steelblue', marker='o', zorder=6.5, linewidth=5, markersize=15)
plt.xticks(rotation=30, ha='right')

ax.set_ylabel('NBE (%)', fontsize=15)
ax.set_xlabel('Irradiance Bin ($W/m^2$)', fontsize=15)
ax.legend([model_name,baseline_model],loc='lower right', fontsize=15)

ax2 = ax.twinx()
ax2 = sns.barplot(x='Irradiance Bins', y='Freq Norm', data=bins, errorbar=None, color='grey', alpha=0.35, zorder=2.5)
ax2.set_ylabel('Binned Irradiance Frequency (%)', fontsize=15)
```

```
plt.grid(False)
plt.xticks(rotation=30, ha='right')
ax.set_zorder(ax2.get_zorder()+1)
ax.patch.set_visible(False)
plt.title('NBE (%) of models in different irradiance bins')
```

Out[24]: Text(0.5, 1.0, 'NBE (%) of models in different irradiance bins')



In [ ]: