Fb-Prophet High Accuracy with irregular data gaps

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("/kaggle/input/solar-power-generation-data/Solar_Power_Generation_data.csv")
df = df.drop('gsp_id',axis=1)
display(df.shape[0])
df = df.rename(columns={'datetime_gmt' : 'DateTime'})
df['DateTime'] = pd.to_datetime(df['DateTime'])
df = df.sort_values(by='DateTime', ascending=True)
df.head()
```

161892

Out[2]:

	DateTime	generation_mw	lcl_mw	ucl_mw	installedcapacity_mwp	capacity_mwp
161891	2015-01-01 00:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5314
161890	2015-01-01 00:30:00+00:00	0.0	0.0	0.0	5453.9499	5386.5314
161889	2015-01-01 01:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313
161888	2015-01-01 01:30:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313
161887	2015-01-01 02:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313

In [3]:
```
df['Year'] = pd.to_datetime(df['DateTime']).dt.year
df['Month'] = pd.to_datetime(df['DateTime']).dt.month
df['DayName'] = pd.to_datetime(df['DateTime']).apply(lambda x: x.day_name())
df['MonthName'] = pd.to_datetime(df['DateTime']).apply(lambda x: x.month_name())
df['Date'] = pd.to_datetime(df['DateTime']).dt.date
df['Quarter'] = pd.to_datetime(df['DateTime']).dt.quarter
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
DateTime                 0
generation_mw           28
lcl_mw                  28
ucl_mw                  28
installedcapacity_mwp    0
capacity_mwp             0
Year                     0
Month                    0
DayName                  0
MonthName                0
Date                     0
Quarter                  0
dtype: int64
```

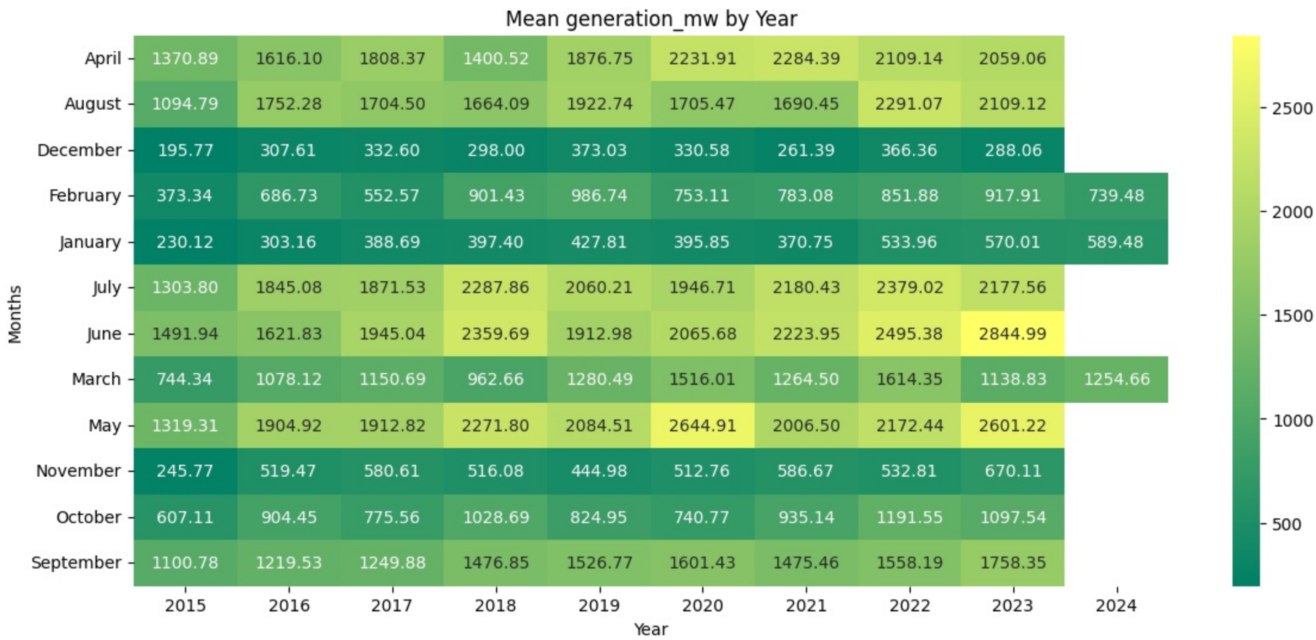
In [5]:
```
df['generation_mw'] = df['generation_mw'].fillna(df['generation_mw'].mean())
df['lcl_mw'] = df['lcl_mw'].fillna(df['lcl_mw'].mean())
df['ucl_mw'] = df['ucl_mw'].fillna(df['ucl_mw'].mean())
df.head()
```

Out[5]:

	DateTime	generation_mw	lcl_mw	ucl_mw	installedcapacity_mwp	capacity_mwp	Year	Month	DayName	MonthName	Date
161891	2015-01-01 00:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5314	2015	1	Thursday	January	2015-01-01
161890	2015-01-01 00:30:00+00:00	0.0	0.0	0.0	5453.9499	5386.5314	2015	1	Thursday	January	2015-01-01
161889	2015-01-01 01:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313	2015	1	Thursday	January	2015-01-01
161888	2015-01-01 01:30:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313	2015	1	Thursday	January	2015-01-01
161887	2015-01-01 02:00:00+00:00	0.0	0.0	0.0	5453.9499	5386.5313	2015	1	Thursday	January	2015-01-01

In [6]:
```
# Mean Price by Month
```

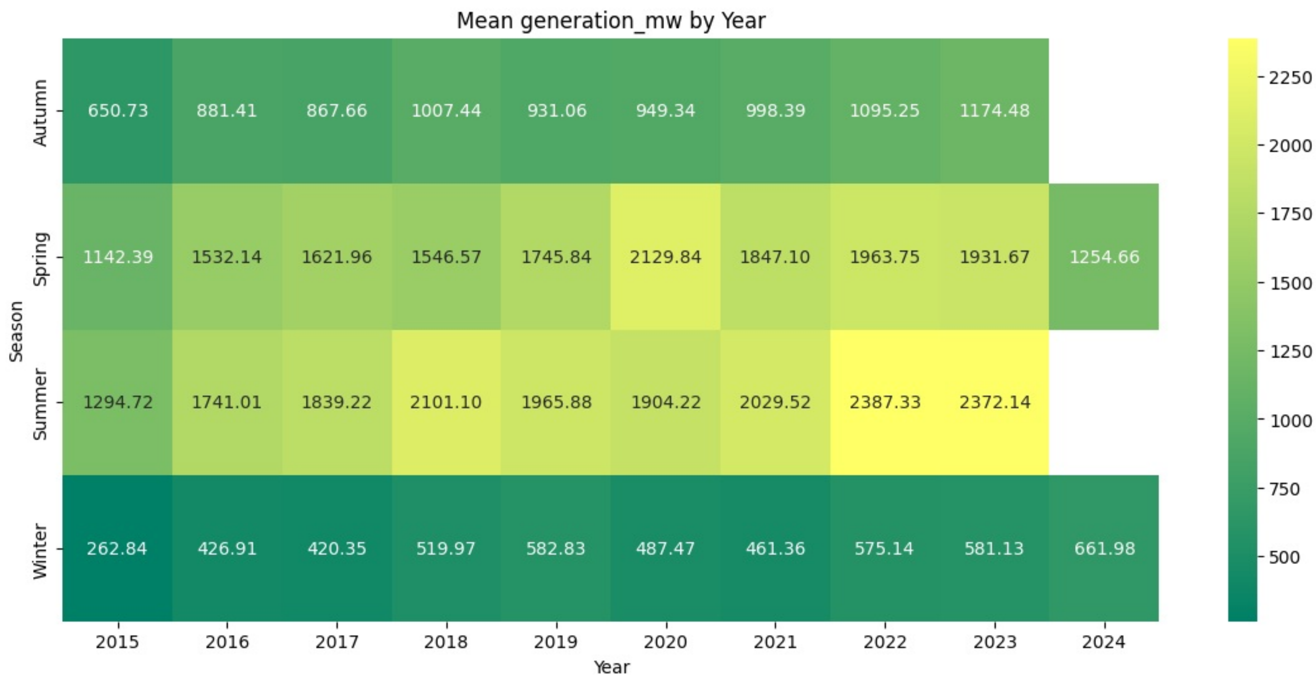
```python
df1 = df.copy('deep')

#  Mean Price by Month
pivot_table = df1.pivot_table(values='generation_mw', index='MonthName', columns='Year', aggfunc='mean')
plt.figure(figsize=(14, 6))
sns.heatmap(pivot_table, annot=True, fmt=".2f", cmap="summer")
plt.title('Mean generation_mw by Year')
plt.xlabel('Year')
plt.ylabel('Months')
plt.show()
```



In [7]:
```python
df["Season"] = [ "Winter" if i < 3 or i > 11 else "Spring" if 3 <= i < 6 else "Summer" if 6 <= i < 9 else "Aut
df['Season'].unique()
```

Out[7]:
```
array(['Winter', 'Spring', 'Summer', 'Autumn'], dtype=object)
```

In [8]:
```python
# Mean Price by Month

df2 = df.copy('deep')

#  Mean Price by Month
pivot_table = df2.pivot_table(values='generation_mw', index='Season', columns='Year', aggfunc='mean')
plt.figure(figsize=(14, 6))
sns.heatmap(pivot_table, annot=True, fmt=".2f", cmap="summer")
plt.title('Mean generation_mw by Year')
plt.xlabel('Year')
plt.ylabel('Season')
plt.show()
```



In [9]:
```python
df3 = df[["Date","generation_mw"]]
df3['Date'] = pd.to_datetime(df3['Date'])
df3.set_index("Date", inplace = True)
```

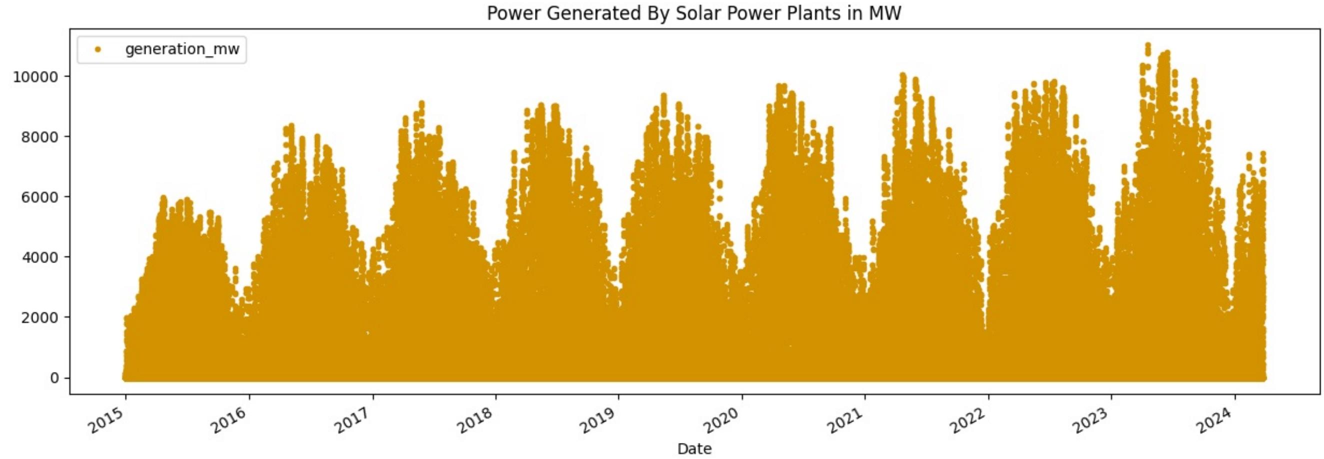
```
df3.head()
```

Out[9]:

	generation_mw
Date	
2015-01-01	0.0
2015-01-01	0.0
2015-01-01	0.0
2015-01-01	0.0
2015-01-01	0.0

In [10]:

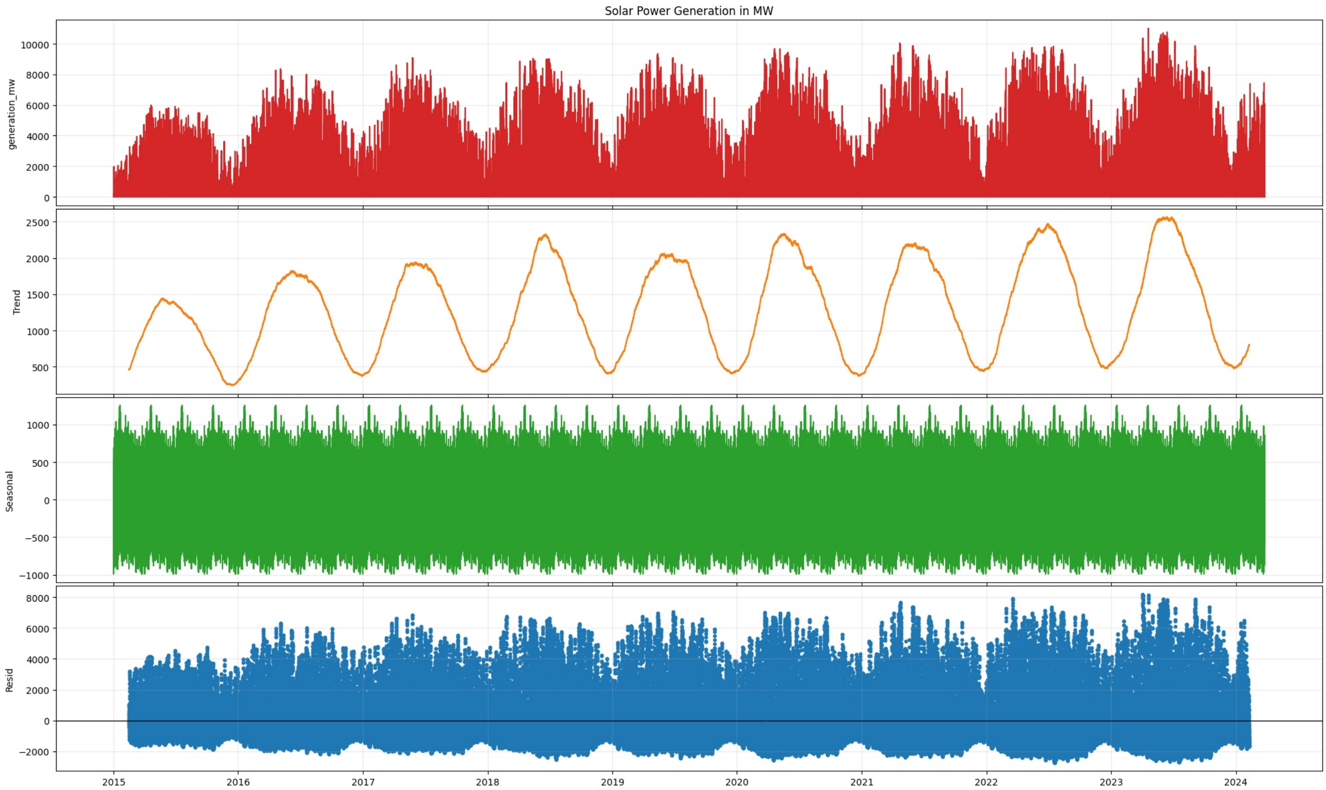
```
# Color pallete for plotting
color_pal = ["#F8766D", "#D39200", "#93AA00","#00BA38", "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
df3.plot(style='.', figsize=(15,5), color=color_pal[1], title='Power Generated By Solar Power Plants in MW')
plt.ylim();
plt.show()
```



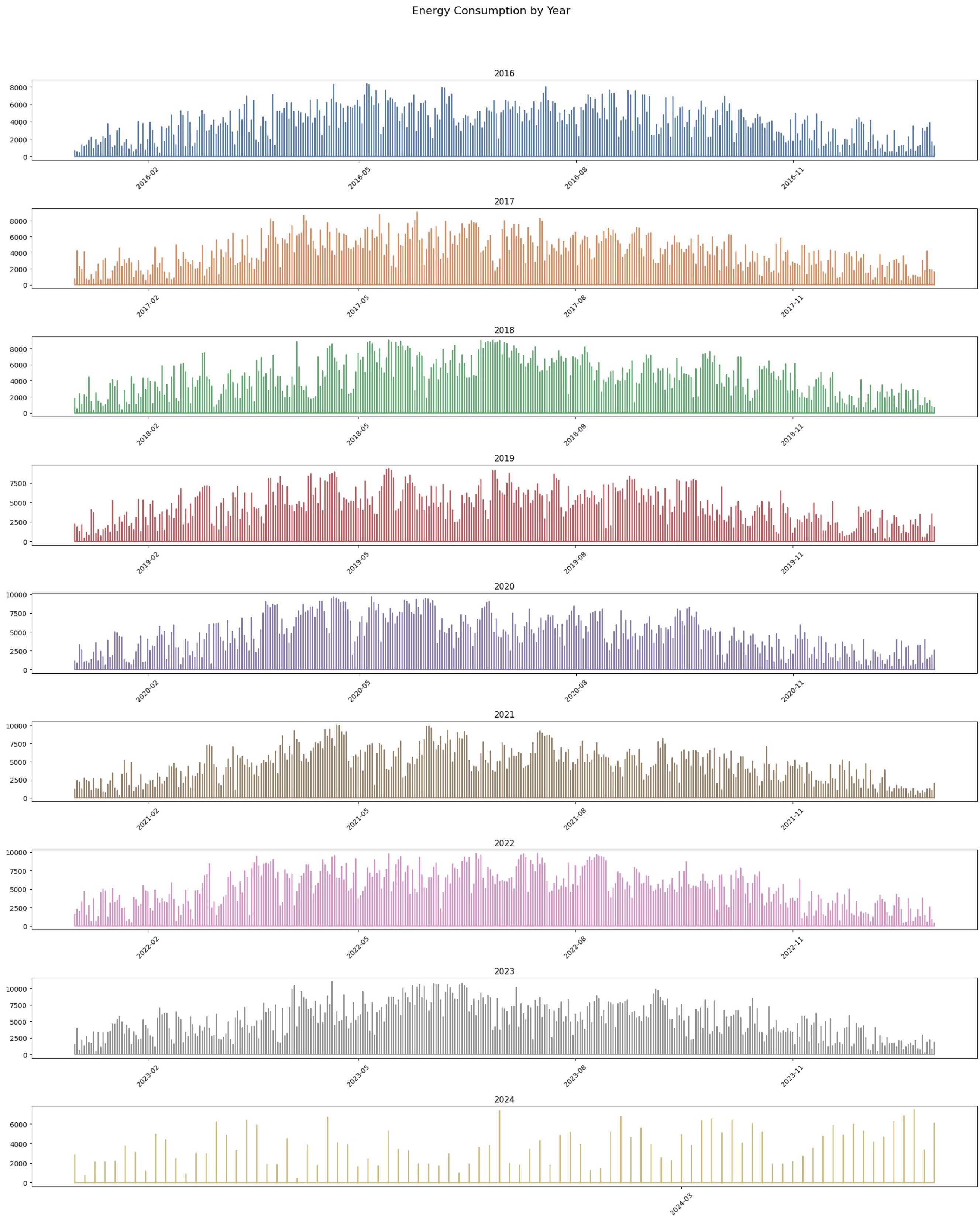
In [11]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

def seasonal_decompose_plotter(df: pd.DataFrame, period=12, title='', figsize=(20, 12)):
    """
    Perform and plot seasonal decomposition of a time series.

    Parameters:
        df: DataFrame with time series data.
        col: Column name for data to decompose. Default is 'sqrt(O3 AQI)'.
        date_col: Column name for datetime values. Default is 'Date'.
        period: Seasonality period. Default is 12.

    Returns:
        A DecomposeResult object with seasonal, trend, and residual components.
    """
    # Decompostion
    decomposition = seasonal_decompose(df.values, period=period)
    de_season = decomposition.seasonal
    de_resid = decomposition.resid
    de_trend = decomposition.trend

    fig, ax = plt.subplots(4, sharex=True, figsize=figsize)

    ax[0].set_title(title)
    ax[0].plot(df.index, df.values, color='C3')
    ax[0].set_ylabel(df.keys()[0])
    ax[0].grid(alpha=0.25)

    ax[1].plot(df.index, de_trend, color='C1')
    ax[1].set_ylabel('Trend')
    ax[1].grid(alpha=0.25)

    ax[2].plot(df.index, de_season, color='C2')
    ax[2].set_ylabel('Seasonal')
    ax[2].grid(alpha=0.25)

    ax[3].axhline(y=0, color='k', linewidth=1)
    ax[3].scatter(df.index, de_resid, color='C0', s=10)
    ax[3].set_ylabel('Resid')
    ax[3].grid(alpha=0.25)

    plt.tight_layout(h_pad=0)
    plt.show()

    return decomposition
```

```
_ = seasonal_decompose_plotter(df3, period=365*12, title='Solar Power Generation in MW', figsize=(20, 12))
```



In [12]:
```
import matplotlib.dates as mdates
year_one = df3.loc[(df3.index >= '2016-01-01') & (df3.index <= '2016-12-31')]
year_two = df3.loc[(df3.index >= '2017-01-01') & (df3.index <= '2017-12-31')]
year_three = df3.loc[(df3.index >= '2018-01-01') & (df3.index <= '2018-12-31')]
year_four = df3.loc[(df3.index >= '2019-01-01') & (df3.index <= '2019-12-31')]
year_five = df3.loc[(df3.index >= '2020-01-01') & (df3.index <= '2020-12-31')]
year_six = df3.loc[(df3.index >= '2021-01-01') & (df3.index <= '2021-12-31')]
year_seven = df3.loc[(df3.index >= '2022-01-01') & (df3.index <= '2022-12-31')]
year_eight = df3.loc[(df3.index >= '2023-01-01') & (df3.index <= '2023-12-31')]
year_nine = df3.loc[(df3.index >= '2024-01-01') & (df3.index <= '2024-12-31')]


years = [year_one, year_two, year_three, year_four,year_five , year_six,year_seven,year_eight,year_nine]
year_labels = ['2016','2017', '2018', '2019', '2020','2021','2022','2023','2024']

fig, axes = plt.subplots(nrows=9, ncols=1, figsize=(20, 26))
fig.suptitle('Energy Consumption by Year', fontsize=16)

axes = axes.flatten()

colors = sns.color_palette('deep', 9)

for i, year in enumerate(years):
    axes[i].plot(year.index, year['generation_mw'], color=colors[i])
    axes[i].set_title(year_labels[i])
    axes[i].set

    axes[i].xaxis.set_major_locator(mdates.MonthLocator(interval=3))  # Setting x axis tick marks/intervals

    axes[i].tick_params(axis='x', rotation=45)  # rotate x-axis labels for better readability

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout
plt.show()
```



Solar Power Generation Forecast with Fb_prophet

In [13]:
```python
df_final = pd.read_csv("/kaggle/input/solar-power-generation-data/Solar_Power_Generation_data.csv")
df_final = df_final.drop('gsp_id',axis=1)
display(df_final.shape[0])
df_final.head()
```

161892

Out[13]:

	datetime_gmt	generation_mw	lcl_mw	ucl_mw	installedcapacity_mwp	capacity_mwp
0	2024-03-26 17:30:00+00:00	421.734	389.983	453.484	16445.201	15438.73
1	2024-03-26 17:00:00+00:00	734.150	694.419	773.880	16445.201	15438.73
2	2024-03-26 16:30:00+00:00	1139.040	1088.380	1189.700	16445.201	15438.73
3	2024-03-26 16:00:00+00:00	1624.860	1560.020	1689.700	16445.201	15438.73
4	2024-03-26 15:30:00+00:00	2075.670	2001.510	2149.820	16445.201	15438.73

In [14]:
```
df_final = df_final[["datetime_gmt","generation_mw"]]
df_final = df_final.sort_values(by='datetime_gmt', ascending=True)
df_final.head()
```

Out[14]:

	datetime_gmt	generation_mw
161891	2015-01-01 00:00:00+00:00	0.0
161890	2015-01-01 00:30:00+00:00	0.0
161889	2015-01-01 01:00:00+00:00	0.0
161888	2015-01-01 01:30:00+00:00	0.0
161887	2015-01-01 02:00:00+00:00	0.0

In [15]:
```
# Format data for prophet model using ds and y
df_final  = df_final.reset_index().rename(columns={'datetime_gmt':'ds','generation_mw':'y'})
df_final = df_final.drop('index',axis=1)
df_final['ds'] = pd.to_datetime(df_final['ds'])
df_final['ds'] = df_final['ds'].dt.tz_localize(None)
df_final.head()
```

Out[15]:

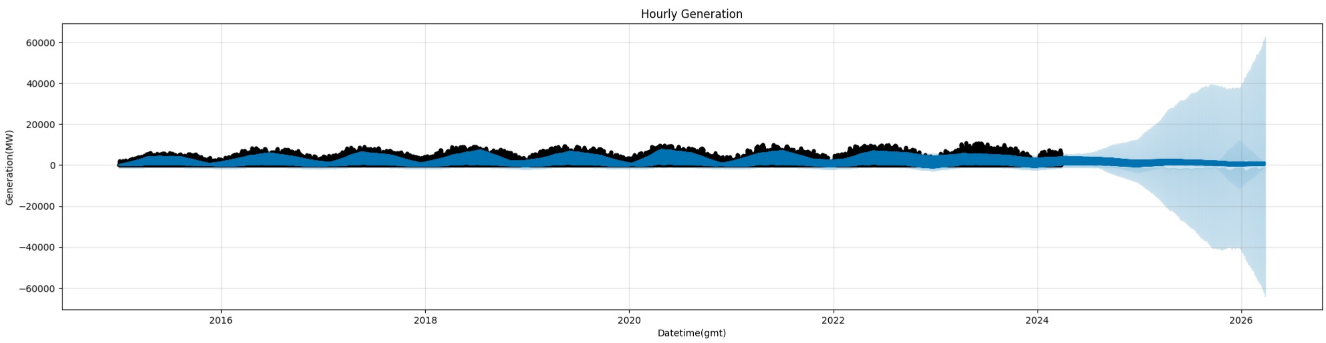
	ds	y
0	2015-01-01 00:00:00	0.0
1	2015-01-01 00:30:00	0.0
2	2015-01-01 01:00:00	0.0
3	2015-01-01 01:30:00	0.0
4	2015-01-01 02:00:00	0.0

In [16]:
```
from prophet import Prophet
## Creating model parameters
model_param ={
    "daily_seasonality": True,
    "weekly_seasonality":True,
    "yearly_seasonality":True,
    "seasonality_mode": "multiplicative",
    "changepoint_prior_scale":0.5
}
```

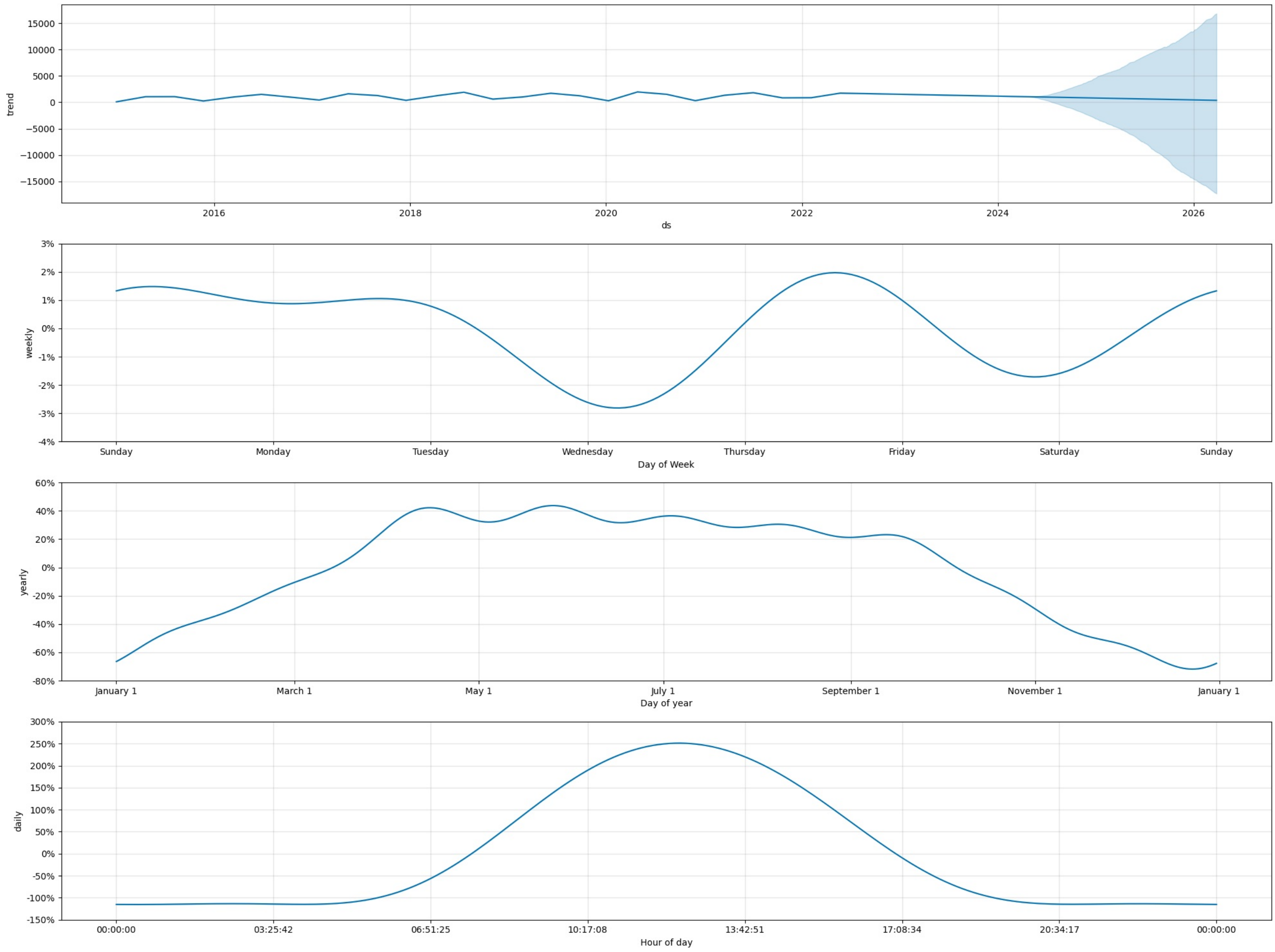
In [17]:
```
model = Prophet(**model_param)
model.fit(df_final)

# Create future dataframe
future= model.make_future_dataframe(periods=365*24*2 ,freq='h')
forecast= model.predict(future)
```

```
11:37:39 - cmdstanpy - INFO - Chain [1] start processing
11:41:11 - cmdstanpy - INFO - Chain [1] done processing
```

In [18]:
```
fig = model.plot(forecast, xlabel='Datetime(gmt)', ylabel=r'Generation(MW)',figsize=(20, 5))
plt.title('Hourly Generation')
plt.show()
```



In [19]:
```
fig3 = model.plot_components(forecast,uncertainty=True,figsize=(20, 15))
plt.show()
```



Data with regular gaps

> ***Problem:***
>
> > 1. Solar power generation data typically has missing values at night (when there's no sunlight).
> > 2. Prophet can handle missing data, but if those gaps occur consistently (e.g., every night), it becomes problematic.
> > 3. During periods with data (daytime), Prophet learns the seasonality (daily cycles).
> > 4. However, during the night gaps, Prophet lacks training data, leading to:
> > 5. Unconstrained seasonality estimates (Prophet can't learn nighttime behavior).
> > 6. Predictions with larger fluctuations than reality (inaccurate nighttime estimations).
>
> Prophet is generally good at handling missing data, it can struggle when those missing values occur in predictable patterns. In the case of solar power generation, nighttime data is often missing entirely. This creates a situation where Prophet learns the daytime seasonality well, but has no data to learn the nighttime behavior. As a result, the model's predictions during nighttime may exhibit much larger fluctuations than what actually happens, leading to less accurate forecasts.

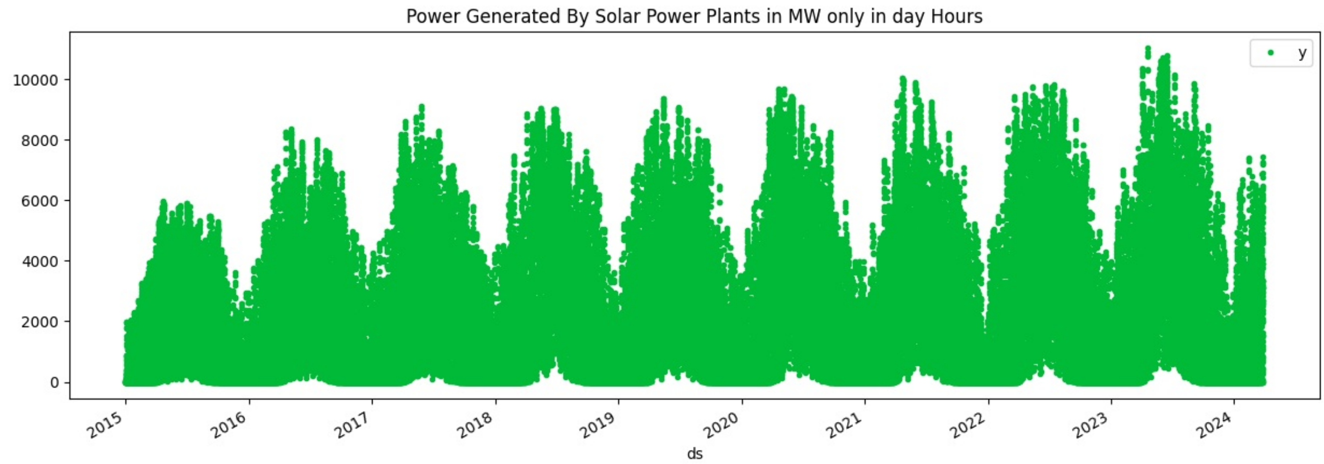
In [20]:
```
df_final = df_final[(df_final['ds'].dt.hour >= 6) & (df_final['ds'].dt.hour < 18)]
df_final.head()
```

Out[20]:

	ds	y
12	2015-01-01 06:00:00	0.0
13	2015-01-01 06:30:00	0.0
14	2015-01-01 07:00:00	0.0
15	2015-01-01 07:30:00	0.0
16	2015-01-01 08:00:00	0.0

In [21]:

```
df5 = df_final[["ds","y"]]
df5.set_index("ds", inplace = True)

# Color pallete for plotting
color_pal = ["#F8766D", "#D39200", "#93AA00","#00BA38", "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
df5.plot(style='.', figsize=(15,5), color=color_pal[3], title='Power Generated By Solar Power Plants in MW onl
plt.ylim();
plt.show()
```



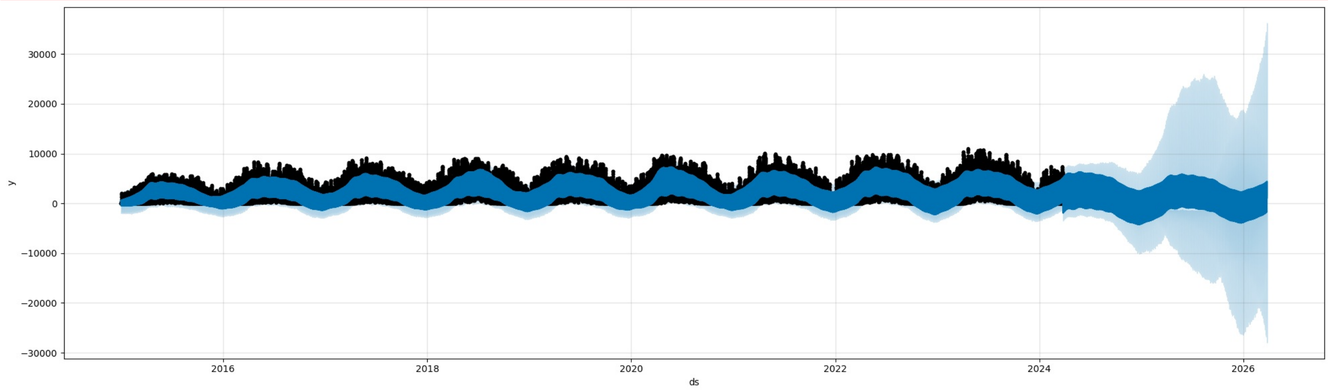
This plot is much sparser than previous plot. We lost all overnight data,

Power Generation is predicted for 2 Years with Hourly data

In [22]:

```
model = Prophet( daily_seasonality=True,
                 weekly_seasonality=True,
                 yearly_seasonality=True,
                 seasonality_mode='multiplicative',
                 changepoint_prior_scale=0.5,
                 holidays_prior_scale=0.1)

model.fit(df_final)

future = model.make_future_dataframe(periods=365*24*2,freq='H')
forecast = model.predict(future)
fig = model.plot(forecast,figsize=(20, 6))
plt.show()
```

```
11:41:55 - cmdstanpy - INFO - Chain [1] start processing
11:43:27 - cmdstanpy - INFO - Chain [1] done processing
```

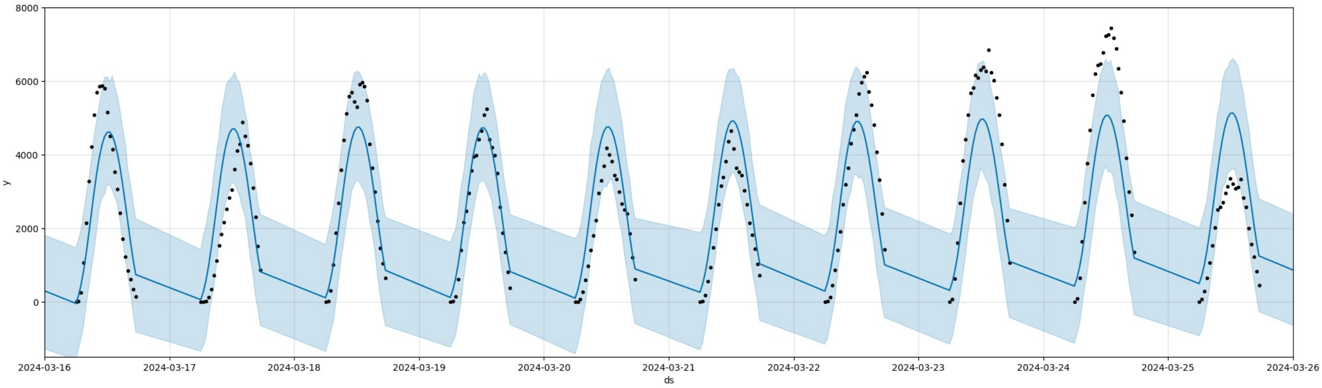


The plotted forecast shows much wider daily fluctuations in the future period than the historical training data

We can zoom in on just 10 days in March 2024 to see more clearly what's

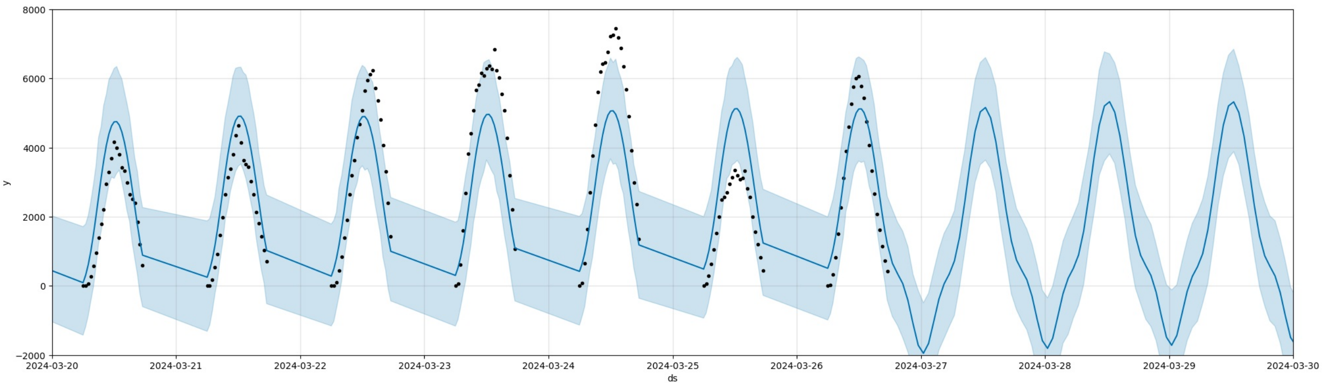
We can zoom in on just 10 days in March 2024 to see more clearly what's going on by replotting and using Matplotlib to constrain the limits of the x and y axes

In [23]:
```
fig = model.plot(forecast,figsize=(20, 6))
plt.xlim(pd.to_datetime(['2024-03-16', '2024-03-26']))
plt.ylim(-1500,8000)
plt.show()
```



****Black dots are actual values and blue line is the forecast****

In [24]:
```
fig = model.plot(forecast,figsize=(20, 6))
plt.xlim(pd.to_datetime(['2024-03-20', '2024-03-30']))
plt.ylim(-2000,8000)
plt.show()
```

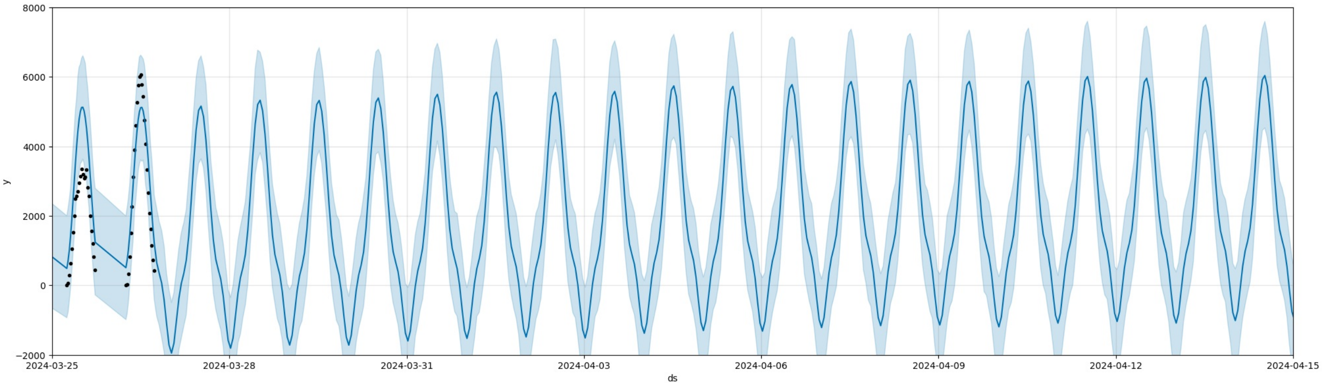


We noticed something interesting with Prophet's predictions for solar power generation. It seems to be predicting a rise in generation specifically on March 27th, 28th, and 29th, 2024. This is likely because Prophet has access to training data from the previous day, which helps it make these specific predictions.

However, there's a catch. Since solar panels don't generate much power before sunrise and after sunset, Prophet struggles in those timeframes. Because it lacks training data for these periods, its predictions tend to be unreliable. It's like trying to guess what happens in the dark – anything's possible, as long as the predictions connect with the known daytime data.

In simpler terms, Prophet shines during the day when it has real data to learn from. But at night, it's flying a bit blind, leading to potentially inaccurate predictions. We need to find a way to improve Prophet's nighttime forecasting abilitie

In [25]:
```
fig = model.plot(forecast,figsize=(20, 6))
plt.xlim(pd.to_datetime(['2024-03-25', '2024-04-15']))
plt.ylim(-2000,8000)
plt.show()
```



fb_Prophet Nighttime Predictions Need Improvement

We saw that Prophet's nighttime predictions were a bit shaky because it lacked training data for those hours. So, instead of trying to force predictions where we don't have good information, here's a clever trick:

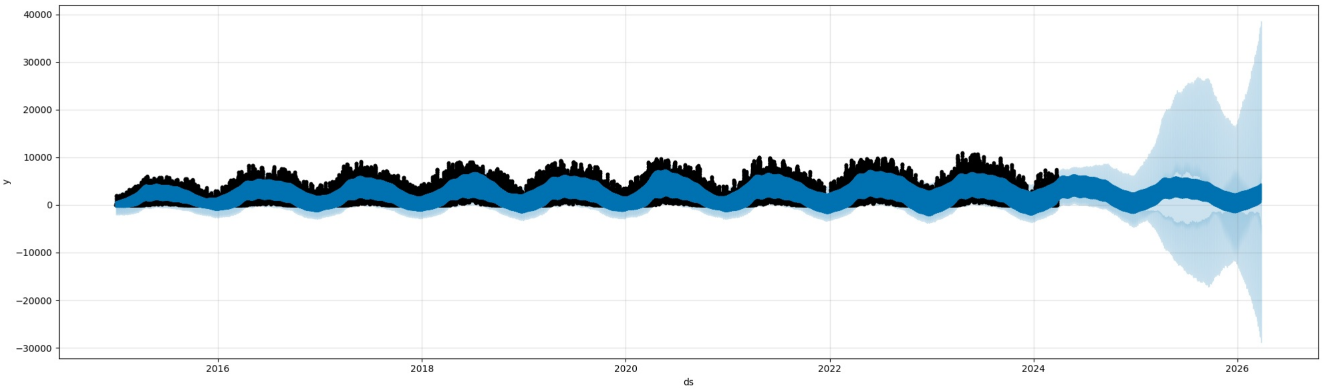
We can simply adjust the timeframe Prophet looks at for predictions! Instead of including the whole day, we can tell it to focus on the hours where we have reliable data - between 6 am and 6 pm. This way, Prophet uses the good daytime data to make predictions, and we avoid the unreliable nighttime guesses.

Think of it like this: We're giving Prophet a smaller, more manageable task. It can focus on the hours where it has the information it needs to make accurate predictions. We don't need to throw the whole day at it and risk getting unreliable results for nighttime.

By adjusting the timeframe for future predictions (like creating a new "future2" DataFrame), we can reuse our existing model and get more accurate forecasts based on the solid daytime data.

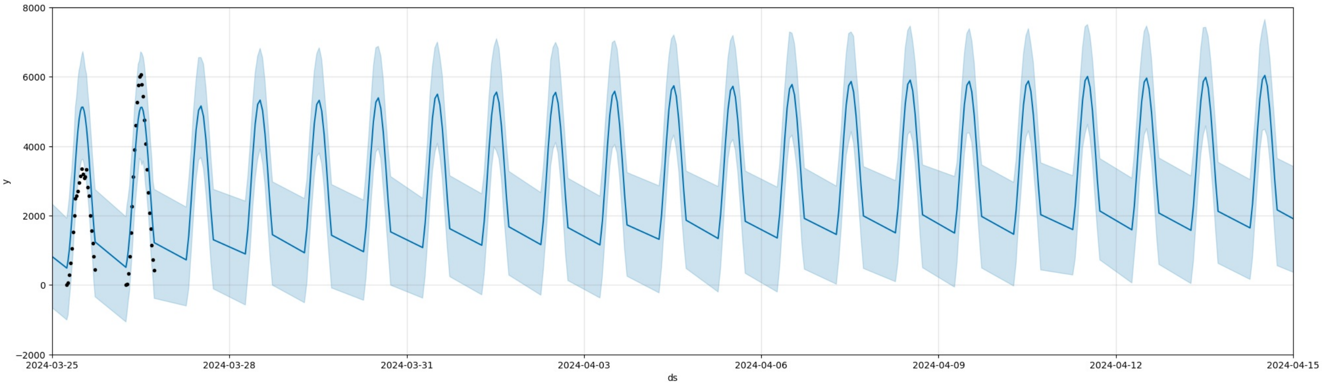
In [26]:
```
future2 = future[(future['ds'].dt.hour >= 6) & (future['ds'].dt.hour < 18)]
```

In [27]:
```
forecast2 = model.predict(future2)
fig = model.plot(forecast2,figsize=(20, 6))
plt.show()
```



The daily fluctuations in the predicted future are of the same magnitude as our historical training data.

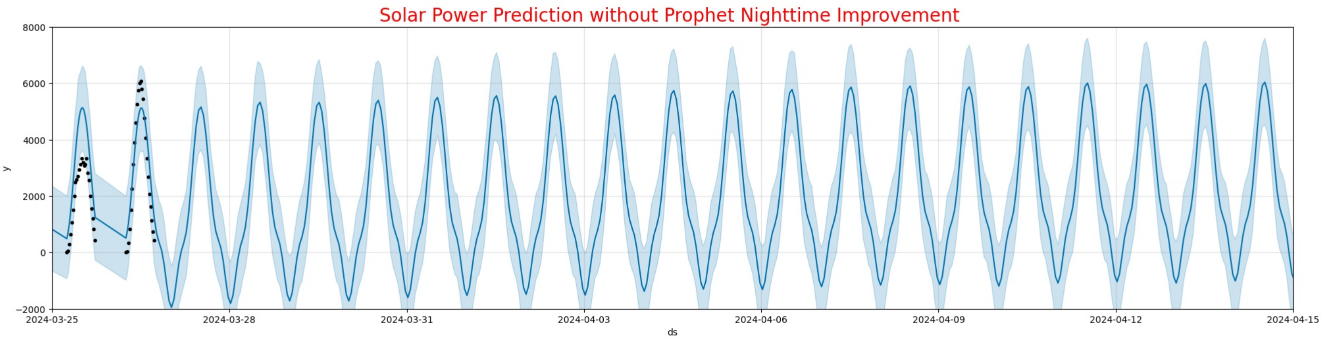
In [28]:
```
fig = model.plot(forecast2,figsize=(20, 6))
plt.xlim(pd.to_datetime(['2024-03-25', '2024-04-15']))
plt.ylim(-2000,8000)
plt.show()
```

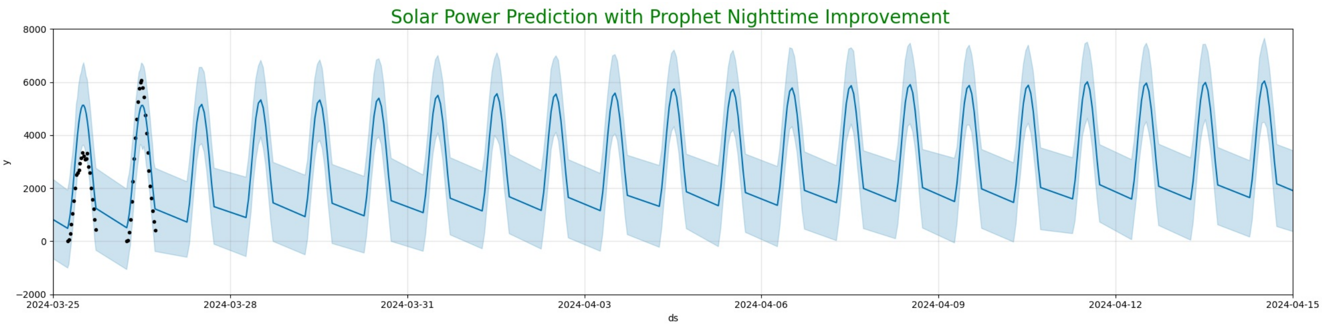


We see the same curve as before for the hours between 6 a.m. and 6 p.m., but this time, Prophet simply connects them with a straight line. There is, in fact, no data in our forecast DataFrame for these time periods; Prophet simply ignores them:

Comparing Solar Data Prediction with and without Prophet Nighttime Improvement

In [29]:
```
fig = model.plot(forecast,figsize=(20, 5))
plt.xlim(pd.to_datetime(['2024-03-25', '2024-04-15']))
plt.ylim(-2000,8000)
plt.title("Solar Power Prediction without Prophet Nighttime Improvement ",fontsize=20,color='Red')
plt.show()

fig = model.plot(forecast2,figsize=(20, 5))
plt.xlim(pd.to_datetime(['2024-03-25', '2024-04-15']))
plt.ylim(-2000,8000)
plt.title("Solar Power Prediction with Prophet Nighttime Improvement ",fontsize=20,color='Green')
plt.tight_layout()
plt.savefig('SolarPowerPrediction.png');
plt.show()
```





https://www.kaggle.com/code/pythonafroz/fb-prophet-high-accuracy-with-irregular-data-gaps

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js