In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
import warnings
import datetime as dt
import matplotlib.dates as mdates
warnings.filterwarnings('ignore')
```

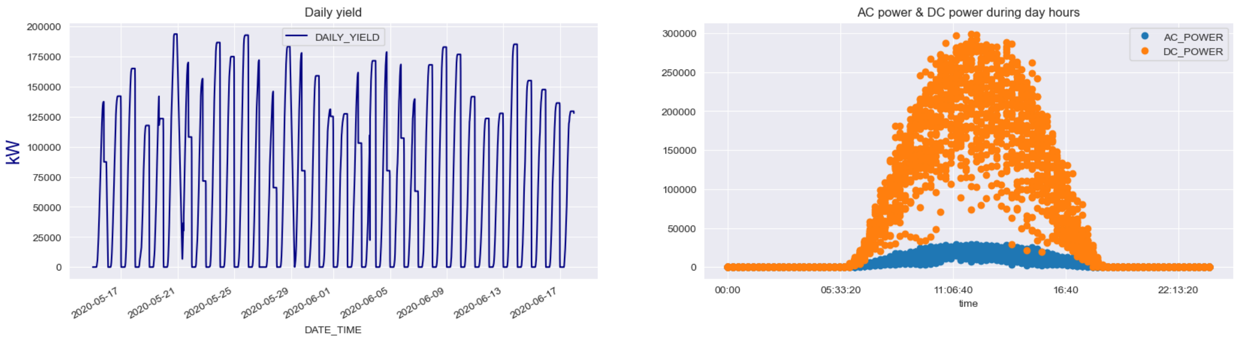
In [2]:

```
# 数据加载与预处理
gen_1 = pd.read_csv('Plant_1_Generation_Data.csv')
gen_1.drop('PLANT_ID', 1, inplace=True)  # 删除PLANT_ID列
sens_1 = pd.read_csv('Plant_1_Weather_Sensor_Data.csv')
sens_1.drop('PLANT_ID', 1, inplace=True)  # 删除PLANT_ID列

# 转换日期时间格式
gen_1['DATE_TIME'] = pd.to_datetime(gen_1['DATE_TIME'], format='%d-%m-%Y %H:%M')
sens_1['DATE_TIME'] = pd.to_datetime(sens_1['DATE_TIME'], format='%Y-%m-%d %H:%M:%S')
```

数据分析

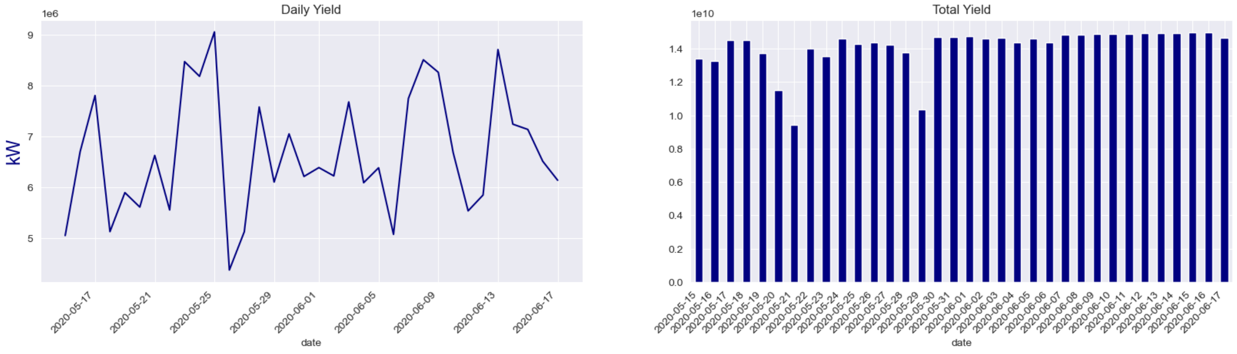
In [3]:

```
# 数据分组与转换
df_gen = gen_1.groupby('DATE_TIME').sum().reset_index()
df_gen['time'] = df_gen['DATE_TIME'].dt.time

# 绘制日发电量图
fig, ax = plt.subplots(ncols=2, nrows=1, dpi=100, figsize=(20, 5))
df_gen.plot(x='DATE_TIME', y='DAILY_YIELD', color='navy', ax=ax[0])

# 绘制交流电功率和直流电功率图
df_gen.set_index('time').drop('DATE_TIME', 1)[['AC_POWER', 'DC_POWER']].plot(style='o', ax=ax[1])
ax[0].set_title('Daily yield')
ax[1].set_title('AC power & DC power during day hours')
ax[0].set_ylabel('kW', color='navy', fontsize=17)
plt.show()
```



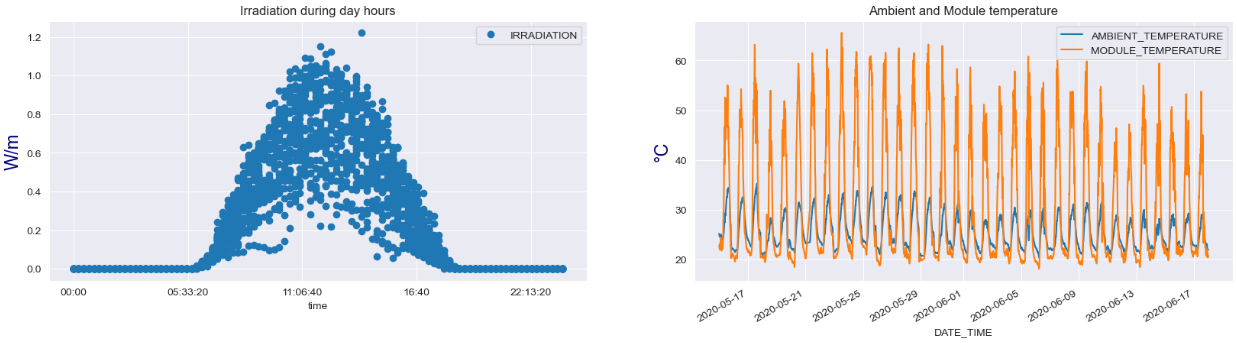
In [4]:

```
# 按日分组，绘制日发电量和总发电量图
daily_gen = df_gen.copy()
daily_gen['date'] = daily_gen['DATE_TIME'].dt.date
daily_gen = daily_gen.groupby('date').sum()

fig, ax = plt.subplots(ncols=2, dpi=100, figsize=(20, 5))
daily_gen['DAILY_YIELD'].plot(ax=ax[0], color='navy')
daily_gen['TOTAL_YIELD'].plot(kind='bar', ax=ax[1], color='navy')

fig.autofmt_xdate(rotation=45)
ax[0].set_title('Daily Yield')
ax[1].set_title('Total Yield')
ax[0].set_ylabel('kW', color='navy', fontsize=17)
plt.show()
```



In [5]:

```
# 数据分组，绘制辐射量和环境/模块温度图
df_sens = sens_1.groupby('DATE_TIME').sum().reset_index()
df_sens['time'] = df_sens['DATE_TIME'].dt.time

fig, ax = plt.subplots(ncols=2, nrows=1, dpi=100, figsize=(20, 5))
df_sens.plot(x='time', y='IRRADIATION', ax=ax[0], style='o')
df_sens.set_index('DATE_TIME').drop('time', 1)[['AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE']].plot(ax=ax[1])

ax[0].set_title('Irradiation during day hours')
ax[1].set_title('Ambient and Module temperature')
ax[0].set_ylabel('W/m', color='navy', fontsize=17)
ax[1].set_ylabel('°C', color='navy', fontsize=17)
plt.show()
```

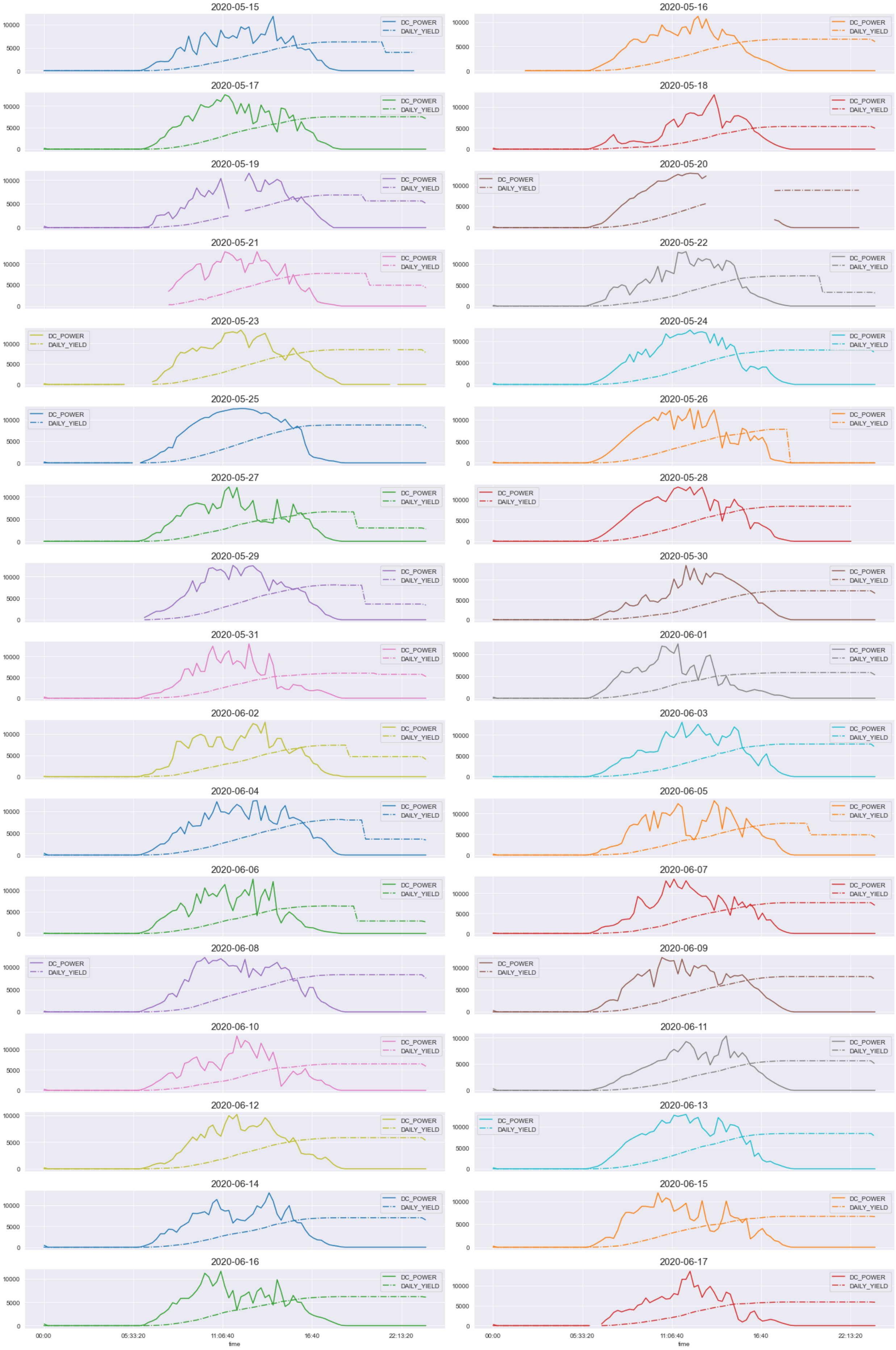


In [7]:

```
# 更详细的时间序列分析
temp1_gen = gen_1.copy()
temp1_gen['time'] = temp1_gen['DATE_TIME'].dt.time
temp1_gen['day'] = temp1_gen['DATE_TIME'].dt.date

temp1_sens = sens_1.copy()
temp1_sens['time'] = temp1_sens['DATE_TIME'].dt.time
temp1_sens['day'] = temp1_sens['DATE_TIME'].dt.date

cols = temp1_gen.groupby(['time', 'day'])['DC_POWER'].mean().unstack()
```

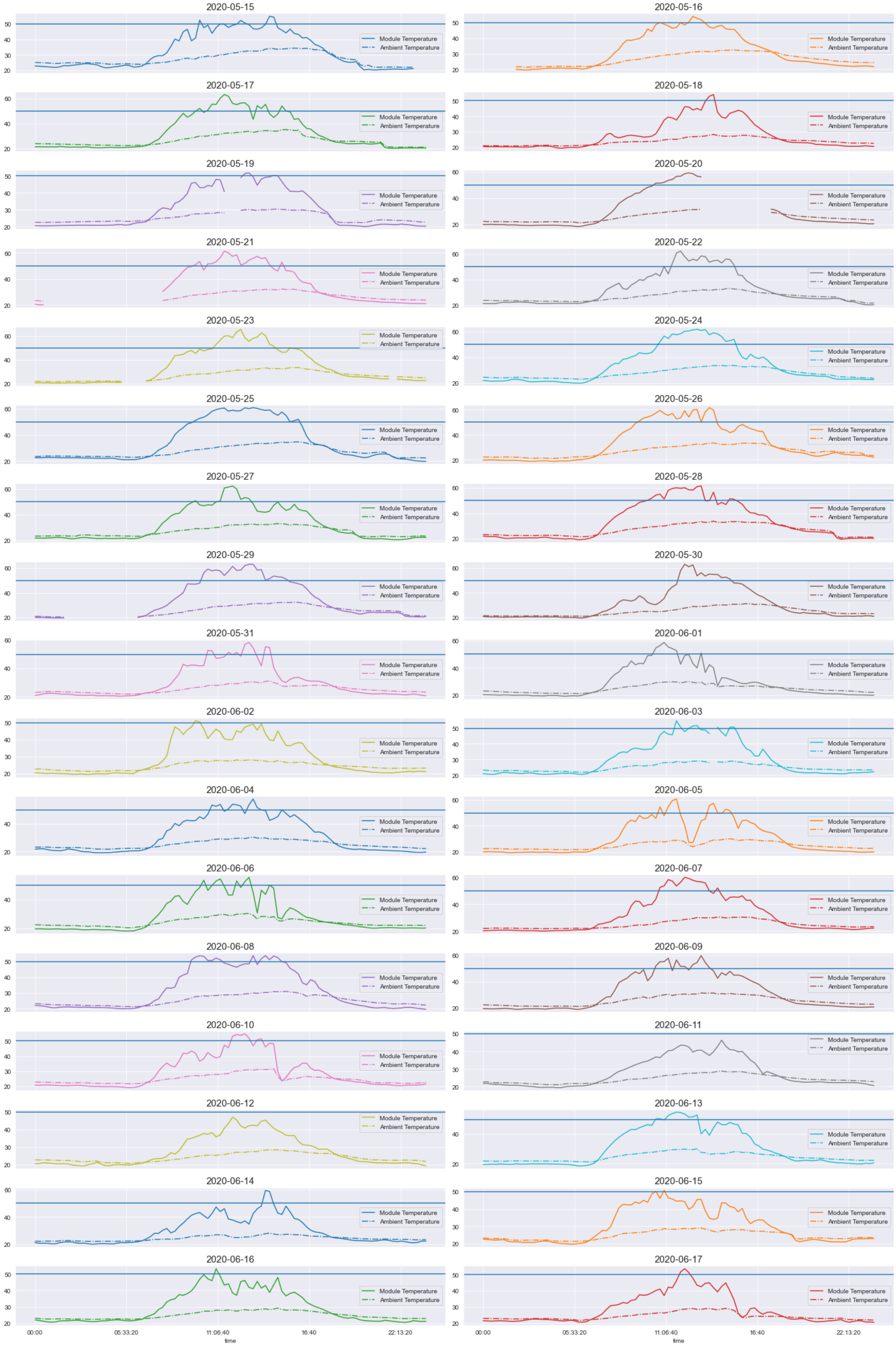
In [8]:

```
# 绘制直流电功率和日发电量图
ax = temp1_gen.groupby(['time', 'day'])['DC_POWER'].mean().unstack().plot(sharex=True, subplots=True, layout
temp1_gen.groupby(['time', 'day'])['DAILY_YIELD'].mean().unstack().plot(sharex=True, subplots=True, layout=(

i = 0
for a in range(len(ax)):
    for b in range(len(ax[a])):
        ax[a, b].set_title(cols.columns[i], size=15)
        ax[a, b].legend(['DC_POWER', 'DAILY_YIELD'])
        i += 1

plt.tight_layout()
plt.show()
```



In [9]:

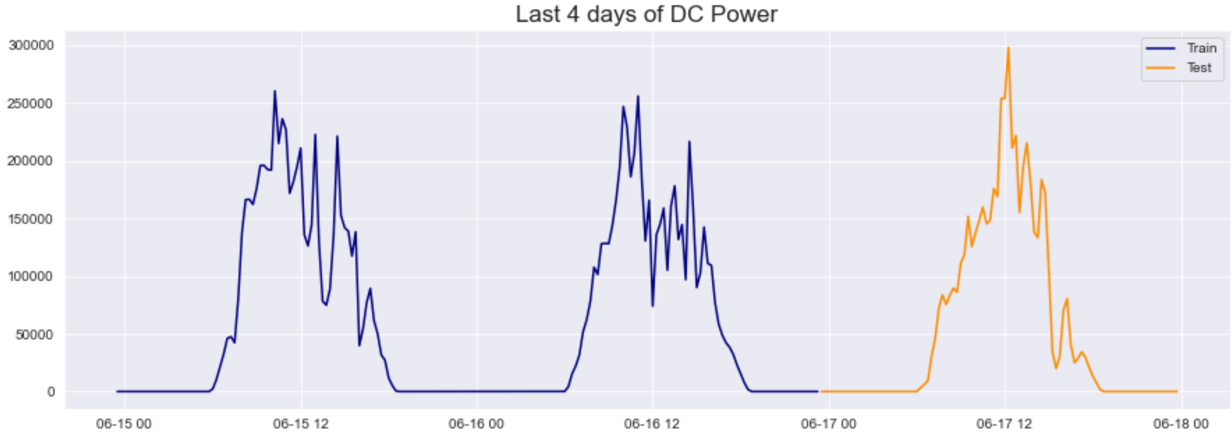
```
# 绘制模块温度和环境温度图
ax = temp1_sens.groupby(['time', 'day'])['MODULE_TEMPERATURE'].mean().unstack().plot(subplots=True, layout=(
temp1_sens.groupby(['time', 'day'])['AMBIENT_TEMPERATURE'].mean().unstack().plot(subplots=True, layout=(17, 2

i = 0
for a in range(len(ax)):
    for b in range(len(ax[a])):
        ax[a, b].axhline(50)
        ax[a, b].set_title(cols.columns[i], size=15)
        ax[a, b].legend(['Module Temperature', 'Ambient Temperature'])
        i += 1

plt.tight_layout()
plt.show()
```



太阳能发电量预测模型

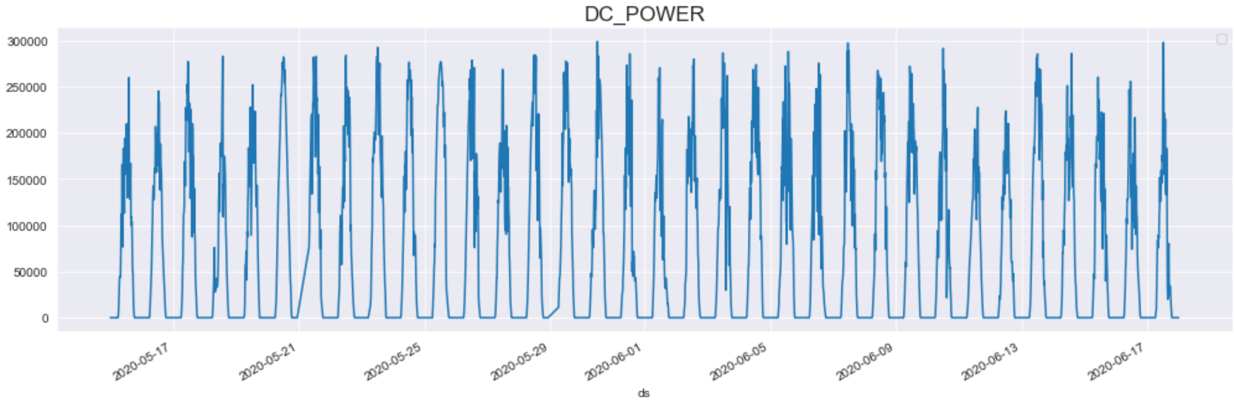
In [57]:

```
# 数据分组，准备预测模型
pred_gen = gen_1.copy()
pred_gen = pred_gen.groupby('DATE_TIME').sum()
pred_gen = pred_gen['DC_POWER'][-288:].reset_index()
pred_gen.set_index('DATE_TIME', inplace=True)
pred_gen.head()

# 分割训练和测试数据集
train = pred_gen[:192]
test = pred_gen[-96:]
plt.figure(figsize=(15, 5))
plt.plot(train, label='Train', color='navy')
plt.plot(test, label='Test', color='darkorange')
plt.title('Last 4 days of DC Power', fontsize=17)
plt.legend()
plt.show()
```



In [58]:

```
import prophet
from prophet import Prophet
import cmdstanpy
cmdstanpy.install_cmdstan(compiler=True)

# 使用 Prophet 对数据进行预测
pred_gen2 = gen_1.copy()
pred_gen2 = pred_gen2.groupby('DATE_TIME')['DC_POWER'].sum().reset_index()
pred_gen2.rename(columns={'DATE_TIME': 'ds', 'DC_POWER': 'y'}, inplace=True)
pred_gen2.plot(x='ds', y='y', figsize=(17, 5))
plt.legend('')
plt.title('DC_POWER', size=17)
plt.show()
```



In [15]:

```
cmdstanpy.install_cmdstan(compiler=True)
```

```
17:42:37 - cmdstanpy - INFO - Add C++ toolchain to $PATH: C:\Users\IvanL\.cmdstan\RTools40
```

```
CmdStan install directory: C:\Users\IvanL\.cmdstan
Installing CmdStan version: 2.33.1
Downloading CmdStan version 2.33.1
Download successful, file: C:\Users\IvanL\AppData\Local\Temp\tmp393irrv2
Extracting distribution
Unpacked download as cmdstan-2.33.1
Building version cmdstan-2.33.1, may take several minutes, depending on your system.
Installed cmdstan-2.33.1
Test model compilation
```

Out[15]:

```
True
```

In [73]:

```python
# 创建和训练 Prophet 模型
m = Prophet()
m.fit(pred_gen2)
```

```
18:27:01 - cmdstanpy - INFO - Chain [1] start processing
18:27:02 - cmdstanpy - INFO - Chain [1] done processing
```

Out[73]:

```
<prophet.forecaster.Prophet at 0x1c260a8f190>
```

In [74]:

```python
# 创建未来时间点用于预测
from pandas.tseries.offsets import DateOffset
future = [pred_gen2['ds'].iloc[-1:] + DateOffset(minutes=x) for x in range(0, 2910, 15)]
time1 = pd.DataFrame(future).reset_index().drop('index', 1)
time1.rename(columns={3157: 'ds'}, inplace=True)
```

In [75]:

```python
timeline = pd.DataFrame(pred_gen2['ds'])
fut = timeline.append(time1, ignore_index=True)
fut.tail()
```

Out[75]:

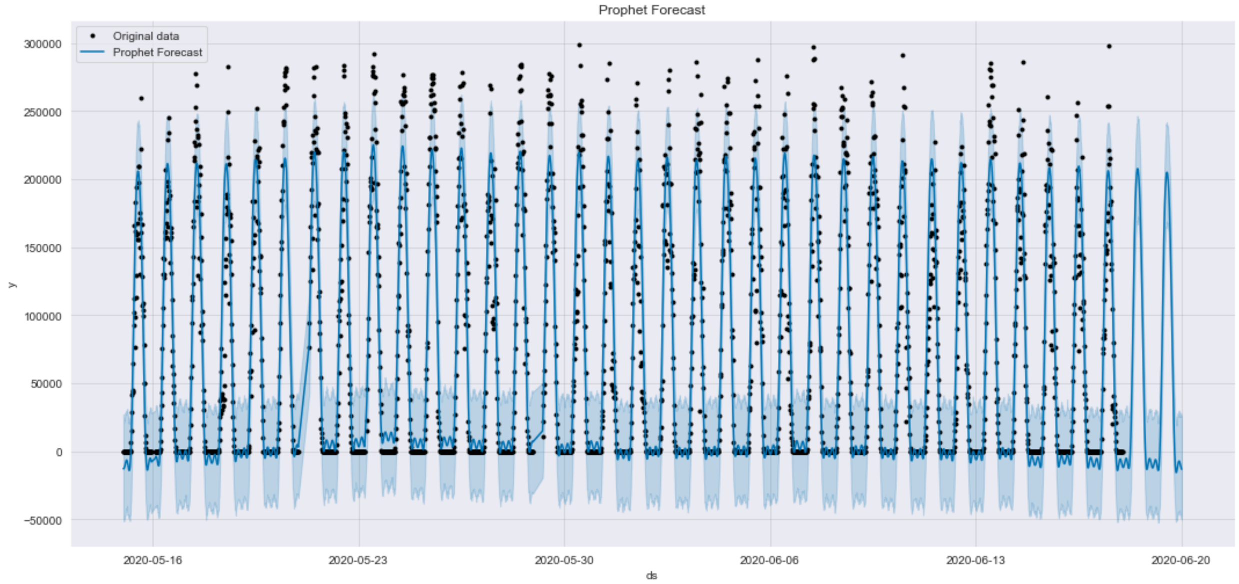
	ds
3347	2020-06-19 23:00:00
3348	2020-06-19 23:15:00
3349	2020-06-19 23:30:00
3350	2020-06-19 23:45:00
3351	2020-06-20 00:00:00

In [76]:

```python
# 进行预测并绘制预测图
forecast = m.predict(fut)
```

In [77]:

```python
m.plot(forecast, figsize=(15, 7))
plt.title('Prophet Forecast')
plt.legend(labels=['Original data', 'Prophet Forecast'])
plt.show()
```



In [84]:

```python
# 使用评估指标计算预测准确性
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
test2 = pd.DataFrame(test.index)
test2.rename(columns={'DATE_TIME': 'ds'}, inplace=True)
test_prophet = m.predict(test2)
```

In [85]:

```
print('Prophet R2 Score: %f' % (r2_score(test['DC_POWER'], test_prophet['yhat'])))
print('-' * 15)
print('Prophet MAE Score: %f' % (mean_absolute_error(test['DC_POWER'], test_prophet['yhat'])))
print('-' * 15)
print('Prophet RMSE Score: %f' % (mean_squared_error(test['DC_POWER'], test_prophet['yhat'], squared=False)))
```

```
Prophet R2 Score: 0.871657
---------------
Prophet MAE Score: 18693.288427
---------------
Prophet RMSE Score: 27502.463672
```

数据储存

In [86]:

```
# 数据采样和后处理
import pandas as pd
forecast_df = pd.read_csv('forecast.csv')
forecast_df['ds'] = pd.to_datetime(forecast_df['ds'])
```

In [89]:

```
# 定义函数以调整和采样数据
def adjust_and_resample(data, start_day):
    data['hour'] = data['ds'].dt.hour + 24 * (data['ds'].dt.day - start_day)
    data = data
    # 采样为每小时平均
    data = data.set_index('ds').resample('H').mean()
    data['hour'] = data.index.hour + 24 * (data.index.day - start_day)
    return data.reset_index()

# 选择特定日期的预测数据进行分析
june_18_data = forecast_df[(forecast_df['ds'].dt.month == 6) &
                           ((forecast_df['ds'].dt.day == 18) |
                            ((forecast_df['ds'].dt.day == 19) & (forecast_df['ds'].dt.hour == 0)))]

june_19_data = forecast_df[(forecast_df['ds'].dt.month == 6) &
                           ((forecast_df['ds'].dt.day == 19) |
                            ((forecast_df['ds'].dt.day == 20) & (forecast_df['ds'].dt.hour == 0)))]

# 调整和采样数据
june_18_data = adjust_and_resample(june_18_data, 18)
june_19_data = adjust_and_resample(june_19_data, 19)

# 合并两天的数据
combined_data = pd.concat([june_18_data, june_19_data]).reset_index(drop=True)

# 确保预测值不为负
combined_data['yhat'] = combined_data['yhat'].clip(lower=0)

# 重组数据，以小时为第一列
combined_data = combined_data[['hour'] + [col for col in combined_data.columns if col != 'hour']]

# 保存合并后的数据到 CSV 文件
combined_file_path = 'combined_hourly_forecast_june_18_19.csv'
combined_data.to_csv(combined_file_path, index=False)
```