IE 360 Statistical Forecasting and Time Series

Spring 2024 – Project

Solar Power Forecasting

Hüseyin Kızıltaş 2020402165

Cem Erdem Yılmaz 2020402090

Ahmet Umut Günbak 2020402129

1. Introduction

Our objective in this study is to accurately forecast the hourly solar power for Edikli GES (Güneş Enerjisi Santrali), which is located at 38.29° North and 34.97° East in Niğde. Although solar energy offers a sustainable energy alternative, its efficiency depends largely on precise projections of future production because of storage constraints and variations in availability throughout the day. Our ability to forecast the hourly solar power output of the following day allows us to improve energy distribution and reduce the risks involved in trading solar power.

We have used hourly weather data from twenty-five locations close to Edikli GES into our research. This information includes a range of meteorological metrics, including temperature, relative humidity, downward shortwave radiation flux, and percentage of total cloud cover for low-level clouds. It is crucial to comprehend these variables because, for example, rising temperatures can lower the voltage outputs from solar panels, and cloud cover or humidity can lower the efficiency of production as a whole.

We performed time series analysis and data visualizations using Python for statistical analysis, including techniques like autoregressive moving average and time series regression. To determine which model is the most accurate for projecting hourly solar power generation at Edikli GES, a number of models will be tested and compared according to their error metrics. By enhancing the accuracy and efficiency of solar power generation forecasts, this strategy hopes to provide improved energy management and planning.

2. Related Literature

Our objective in this study is to accurately forecast the hourly solar power for Edikli GES (Güneş Enerjisi Santrali), which is located at 38.29° North and 34.97° East in Niğde. Although solar energy offers a sustainable energy alternative, its efficiency depends largely on precise projections of future production because of storage constraints and variations in availability throughout the day. Our ability to forecast the hourly solar power output of the following day allows us to improve energy distribution and reduce the risks involved in trading solar power.

We have used hourly weather data from twenty-five locations close to Edikli GES into our research. This information includes a range of meteorological metrics, including temperature, relative humidity, downward shortwave radiation flux, and percentage of total cloud cover for low-level clouds. It is crucial to comprehend these variables because, for example, rising temperatures can lower the voltage outputs from solar panels, and cloud cover or humidity can lower the efficiency of production as a whole.

We performed time series analysis and data visualizations using Python for statistical analysis, including techniques like autoregressive moving average and time series regression. To determine which model is the most accurate for projecting hourly solar power generation at Edikli GES, a number of models will be tested and compared according to their error metrics. By enhancing the accuracy and efficiency of solar power generation forecasts, this strategy hopes to provide improved energy management and planning.

3. Approach

Our approach to forecasting solar power production for Edikli GES involved a progressive refinement of methodologies, starting with simpler models and ultimately arriving at a more sophisticated, data-driven solution using ARIMA. The progression of our approach is detailed below:

Initial Naive Approach Initially, we attempted to predict solar power production by averaging the weather data from all 25 coordinate points. This naive approach, however, neglected the unique influence of each coordinate on the power plant's production, resulting in inaccurate predictions.

Random Forest Regression Models To improve accuracy, we explored Random Forest Regression models. This involved several iterations and refinements:

Basic Random Forest Regression:

- **Data Preparation:** We merged the production data with weather data and engineered additional features, such as the day of the

week.

- **Model Training:** Separate Random Forest models were trained for each hour of the day. This accounted for the diurnal patterns in solar power production.
- **Evaluation:** Although this approach provided better results than the naive method, it still lacked the precision needed for accurate forecasting.

Advanced Random Forest with Hyperparameter Tuning:

- **Enhanced Feature Engineering:** We added lag features (e.g., production from the previous day and previous week) to capture temporal dependencies.
- **Model Optimization:** Hyperparameter tuning was performed using Grid Search with TimeSeriesSplit cross-validation, optimizing parameters such as the number of trees, maximum depth, and other model-specific parameters.
- **Model Saving and Loading:** Models were saved using joblib for efficient storage and loading during the prediction phase.

Despite improvements, the Random Forest models occasionally predicted non-physical values, such as negative production, and failed to adequately capture the nighttime zero production.

Transition to ARIMA Models Recognizing the limitations of Random Forest models in capturing time-dependent patterns accurately, we transitioned to ARIMA models. ARIMA, being inherently suited for time series data, offered a more robust framework for our forecasting needs.

Data Preparation:

- **Weather Data Weighting:** Each weather variable from the 25 coordinates was weighted based on its correlation with solar power production. This ensured that the unique influence of each coordinate was accounted for.
- **Feature Engineering:** Weighted weather data were combined, and additional features like the day of the week were included.

Hourly SARIMAX Models:

- **Separate Hourly Models:** ARIMA models were trained separately for each hour of the day. This approach allowed us to capture the specific temporal patterns in solar power production for each hour.
- **Automatic Parameter Selection:** The auto_arima function from the pmdarima library was used to automatically select the best p, d, and q parameters, optimizing model performance without overburdening computational resources.

Model Training and Forecasting: The ARIMA models were trained on the merged production and weighted weather data. Predictions for the next day were made using the trained models, ensuring accurate and non-negative forecasts.

Performance Evaluation The models were evaluated using standard metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). The transition to ARIMA models significantly improved the accuracy of our predictions, addressing the shortcomings of the initial Random Forest models and ensuring realistic production forecasts.

In [ ]:

```
import requests
import zipfile
import os
from io import BytesIO

def download_and_extract_zip(zip_url, extract_path):
    """Downloads and extracts a ZIP file from Google Drive to the specified path."""

    try:
        response = requests.get(zip_url)
        response.raise_for_status()  # Raise an error for bad responses

        with zipfile.ZipFile(BytesIO(response.content)) as zip_file:
            for member in zip_file.namelist():
                filename = os.path.basename(member)
                if not filename:
                    continue  # Skip directories

                source = zip_file.open(member)
                target_path = os.path.join(extract_path, filename)

                with open(target_path, "wb") as target_file:
                    target_file.write(source.read())

    except requests.exceptions.RequestException as e:
        print(f"Download error for '{zip_url}': {e}")
    except zipfile.BadZipFile:
        print(f"Invalid ZIP file: '{zip_url}'.")
    except Exception as e:
        print(f"Unexpected error for '{zip_url}': {e}")
```

```
# List of Google Drive file IDs
zip_file_ids = ["1As-67MFNrpim_jGYgSOgT2SF119gP790", "1pjaRe_lSIygHyZ8luWplSC6fuwAEmwTd"]

# Get the current working directory
script_directory = os.getcwd()

# Construct direct download links for the ZIP files
zip_urls = [f"https://drive.google.com/uc?export=download&id={file_id}" for file_id in
zip_file_ids]

# Download and extract each ZIP file directly into the current working directory
for zip_url in zip_urls:
    download_and_extract_zip(zip_url, script_directory)
```

1. **Data Loading and Preparation**

Two CSV files named production_data and weather_data are loaded. The date columns are converted to datetime objects. The data is sorted based on the date and hour columns. Missing values in weather_data are filled using data from the previous day with the same hour, latitude, and longitude. 2. **Determining Common Time Periods** The common dates and hours present in both production_data and weather_data are identified. 3. **Separating Data by Coordinates** For each unique latitude and longitude combination in weather_data, a separate CSV file is created: Weather data for each coordinate is filtered. The file name is created in the format koordinat_{latitude}_{longitude}.csv. If the file exists, data is appended; otherwise, the file is created and data is saved. The lat and lon columns are removed. 4. **Correlation Analysis for Coordinates** For each coordinate file: Coordinate data is loaded. It's merged with production_data based on common dates and hours. Correlation between production and each weather variable is calculated. Correlations are stored in a dictionary. 5. **Calculating Coordinate Influence Factors** For each coordinate, the influence factor for each variable is calculated. The influence factor is the square of the absolute value of the correlation. Influence factors are normalized so that the sum of influences for each variable across all coordinates equals 1. Influence factors are converted to a DataFrame and printed to the console. 6. **Calculating Weighted Weather Data** For each coordinate file: Coordinate data is loaded. Each variable is multiplied by the corresponding coordinate's influence factor. Data is grouped by the date and hour columns, and values are summed. The resulting data is appended to a sonuc_df DataFrame. 7. **Saving Weighted Weather Data** The sonuc_df is grouped by date and hour, and values are summed. Values with more than three decimal places are rounded to three decimal places. The entire result is saved to a file named agirlikli.csv. 8. **Cleaning Up** The coordinate files are no longer needed, so they are deleted. This code is essential for incorporating the impact of location-specific weather data into a solar power production forecasting model. The weighted weather data generated by this code will be used as input for further analysis and forecasting.

In [ ]:

```
import pandas as pd
import os
import glob

# Read CSV files
production_data = pd.read_csv("production.csv")
production_data['date'] = pd.to_datetime(production_data['date'])

weather_data = pd.read_csv("processed_weather.csv")
weather_data['date'] = pd.to_datetime(weather_data['date'])

# Sort the data by date and hour
production_data = production_data.sort_values(by=["date", "hour"])
weather_data = weather_data.sort_values(by=["date", "hour", "lat", "lon"])

# Fill missing values in weather_data using the previous day's same hour, lat, and lon values
for col in weather_data.columns:
    if col in ['date', 'hour', 'lat', 'lon']:
        continue
    weather_data[col] = weather_data.groupby(['hour', 'lat', 'lon'])[col].transform(lambda x:
x.fillna(method='ffill'))

# Identify common dates and hours present in both datasets
common_dates = set(production_data["date"].unique()) & set(weather_data["date"].unique())
common_hours = set(production_data["hour"].unique()) & set(weather_data["hour"].unique())

# Save each coordinate's weather data to a separate file
for lat in weather_data["lat"].unique():
    for lon in weather_data["lon"].unique():
        # Filter weather data for the specific coordinate
        coord_weather = weather_data[(weather_data["lat"] == lat) & (weather_data["lon"] ==
lon)]

        # Create the file name
        dosya_adi = f"koordinat_{lat}_{lon}.csv"
```

```
        # Write headers if the file does not exist, otherwise append data without headers
        if not os.path.exists(dosya_adi):
            coord_weather.to_csv(dosya_adi, index=False, header=True)  # Write headers for the
first time
        else:
            coord_weather.to_csv(dosya_adi, mode='a', header=False, index=False)  # Do not
write headers for subsequent appends

        # Open the file and remove lat and lon columns
        df = pd.read_csv(dosya_adi)
        df = df.drop(["lat", "lon"], axis=1)
        df.to_csv(dosya_adi, index=False)

# Perform correlation analysis for each coordinate
koordinat_korelasyonlari = {}
for dosya_adi in glob.glob("koordinat_*.csv"):
    # Read the coordinate's data
    koordinat_data = pd.read_csv(dosya_adi)
    koordinat_data['date'] = pd.to_datetime(koordinat_data['date'])

    # Filter data for common dates and hours
    koordinat_data = koordinat_data[(koordinat_data["date"].isin(common_dates)) &
(koordinat_data["hour"].isin(common_hours))]

    # Merge production data with coordinate data
    merged_data = pd.merge(
        production_data[(production_data["date"].isin(common_dates)) &
(production_data["hour"].isin(common_hours))],
        koordinat_data,
        on=["date", "hour"]
    )

    # Calculate correlation for each weather variable
    correlations = {}
    for col in koordinat_data.columns:
        if col in ['date', 'hour']:
            continue
        correlation = merged_data["production"].corr(merged_data[col])
        correlations[col] = correlation

    # Save correlations for the coordinate
    koordinat_korelasyonlari[dosya_adi] = correlations

# Calculate influence factors for each variable for each coordinate
koordinat_etkileri = {}
for koordinat, korelasyonlar in koordinat_korelasyonlari.items():
    koordinat_etkileri[koordinat] = {}
    for degisken, korelasyon in korelasyonlar.items():
        # Influence is calculated as the square of the absolute value of the correlation
        etki = abs(korelasyon) ** 2
        koordinat_etkileri[koordinat][degisken] = etki

# Normalize influence factors for each variable
for degisken in koordinat_etkileri[list(koordinat_etkileri.keys())[0]].keys():
    etki_toplami = sum(koordinat_etkileri[koordinat][degisken] for koordinat in
koordinat_etkileri)
    for koordinat in koordinat_etkileri:
        koordinat_etkileri[koordinat][degisken] /= etki_toplami

# Convert influence factors to a DataFrame and print them
etki_df = pd.DataFrame(koordinat_etkileri).T.reset_index()
print("Koordinat Etki Çarpanları:")
print(etki_df.to_string(index=False))

# Multiply each column in coordinate files by the corresponding influence factor and combine
the results
sonuc_df = pd.DataFrame()
for dosya_adi in glob.glob("koordinat_*.csv"):
    # Read the coordinate data
    koordinat_data = pd.read_csv(dosya_adi)
    koordinat_data['date'] = pd.to_datetime(koordinat_data['date'])

    # Get the influence factors for the coordinate
    etkiler = koordinat_etkileri[dosya_adi]

    # Multiply each column by the corresponding influence factor
    for degisken, etki in etkiler.items():
        koordinat_data[degisken] = koordinat_data[degisken] * etki
```

```
    # Group by date and hour and sum the values
    koordinat_data = koordinat_data.groupby(['date', 'hour']).sum().reset_index()

    # Append to the result DataFrame
    sonuc_df = pd.concat([sonuc_df, koordinat_data], ignore_index=True)

# Group the results by date and hour and sum the values
sonuc_df = sonuc_df.groupby(['date', 'hour']).sum().reset_index()

# Round values to 3 decimal places where necessary
for col in sonuc_df.columns:
    if col in ['date', 'hour']:
        continue
    sonuc_df[col] = sonuc_df[col].round(3)

# Save the final result to a CSV file
sonuc_df.to_csv("agirlikli.csv", index=False)

# Remove the coordinate files as they are no longer needed
for dosya_adi in glob.glob("koordinat_*.csv"):
    os.remove(dosya_adi)

Koordinat Etki Çarpanları:
                   index  dswrf_surface  tcdc_low.cloud.layer  tcdc_middle.cloud.layer
tcdc_high.cloud.layer  tcdc_entire.atmosphere  uswrf_top_of_atmosphere  csnow_surface
dlwrf_surface  uswrf_surface  tmp_surface
 koordinat_37.75_34.5.csv       0.039208              0.051116                 0.047609
0.038598                0.047854                0.043760       0.013131       0.068287
0.046848     0.040919
koordinat_37.75_34.75.csv       0.039656              0.043970                 0.045240
0.039460                0.043225                0.038579       0.046847       0.030161
0.040910     0.040595
 koordinat_37.75_35.0.csv       0.040042              0.045986                 0.042667
0.040444                0.043662                0.041260       0.067138       0.004902
0.040216     0.042601
koordinat_37.75_35.25.csv       0.040943              0.046307                 0.035072
0.041069                0.038771                0.040159       0.073751       0.001071
0.043755     0.042820
 koordinat_37.75_35.5.csv       0.040115              0.041703                 0.032541
0.041696                0.036987                0.025741       0.030808       0.047095
0.045295     0.042311
  koordinat_38.0_34.5.csv       0.039575              0.046044                 0.047612
0.040947                0.045198                0.039649       0.043513       0.021121
0.038917     0.039196
 koordinat_38.0_34.75.csv       0.039633              0.041420                 0.044655
0.039163                0.042528                0.041646       0.049222       0.029545
0.040490     0.039791
  koordinat_38.0_35.0.csv       0.040494              0.036846                 0.043822
0.039927                0.043207                0.041536       0.059226       0.003093
0.039652     0.039867
 koordinat_38.0_35.25.csv       0.041740              0.036753                 0.040322
0.038268                0.041672                0.040904       0.076498       0.000685
0.036717     0.038696
  koordinat_38.0_35.5.csv       0.040072              0.041976                 0.033846
0.037467                0.037261                0.037561       0.049267       0.007389
0.033629     0.038127
 koordinat_38.25_34.5.csv       0.039848              0.038901                 0.038212
0.037545                0.039552                0.037278       0.046430       0.027947
0.033357     0.041340
koordinat_38.25_34.75.csv       0.039962              0.034660                 0.039739
0.040352                0.037813                0.039772       0.025800       0.068394
0.034939     0.040364
 koordinat_38.25_35.0.csv       0.040185              0.038573                 0.044338
0.041467                0.042061                0.040630       0.023790       0.054023
0.040431     0.040243
koordinat_38.25_35.25.csv       0.040115              0.032396                 0.039443
0.044447                0.041274                0.041749       0.017576       0.079700
0.043466     0.038174
 koordinat_38.25_35.5.csv       0.039955              0.031200                 0.039855
0.042093                0.036661                0.042978       0.032874       0.079325
0.038524     0.039154
  koordinat_38.5_34.5.csv       0.038813              0.037254                 0.035325
0.037762                0.035670                0.038826       0.030430       0.045917
0.033888     0.038568
 koordinat_38.5_34.75.csv       0.039611              0.036704                 0.038002
0.040359                0.037461                0.039380       0.038327       0.032746
0.035018     0.038268
  koordinat_38.5_35.0.csv       0.040231              0.036344                 0.040762
0.039887                0.038896                0.039940       0.033341       0.028899
```

```
0.040215     0.038985
 koordinat_38.5_35.25.csv       0.040208             0.040741                0.042810
0.042860                0.041849                0.043255       0.037621       0.046550
0.041682     0.040290
  koordinat_38.5_35.5.csv       0.040903             0.039545                0.039496
0.040918                0.038377                0.042488       0.064820       0.005720
0.032652     0.036989
 koordinat_38.75_34.5.csv       0.038558             0.046218                0.034753
0.034827                0.036886                0.042328       0.023581       0.063930
0.043639     0.039435
koordinat_38.75_34.75.csv       0.039310             0.045512                0.035092
0.037206                0.038215                0.041056       0.027961       0.057044
0.044500     0.039947
 koordinat_38.75_35.0.csv       0.039906             0.042443                0.039999
0.039757                0.039717                0.041443       0.031301       0.052091
0.043154     0.039761
koordinat_38.75_35.25.csv       0.040266             0.036804                0.040108
0.041371                0.038584                0.039818       0.028157       0.058006
0.041680     0.040814
 koordinat_38.75_35.5.csv       0.040653             0.030584                0.038679
0.042112                0.036620                0.038262       0.028588       0.086359
0.046427     0.042745
```

4. Results:

Descriptive Analysis of Production and Weather Data

In this section, we conduct a detailed analysis of the solar power production and weather data to identify patterns and seasonality that inform our forecasting model.

1. • **Daily Solar Power Production Averages**

We start by analyzing the daily average solar power production to understand the overall trend and seasonality in the production data.

Data Preparation: We load the production data and combine the date and hour columns into a single datetime column. By grouping the data by date, we calculate the average daily production.

Plotting and Observations: The plot of daily average production shows the average daily solar power production over time. There is a clear seasonality in the data, with higher production in the summer months and lower production in the winter months. This pattern reflects the increased solar radiation available during summer.

1. • **Hourly Solar Power Production Averages**

Next, we analyze the average hourly solar power production to understand the diurnal pattern of solar power generation.

Data Preparation: Similar to the daily analysis, we use the combined datetime column. By grouping the data by hour, we calculate the average production for each hour of the day.

Plotting and Observations: The plot shows the average production for each hour of the day. The hourly solar power production resembles a normal distribution, reaching a maximum at mid-day and decreasing towards the early morning and late afternoon hours. This pattern is consistent with the daily cycle of sunlight.

1. • **Weather Data Analysis**

We analyze various weather variables that affect solar power production, using the weighted averages from multiple coordinates.

Variables Analyzed: The weather variables include downward shortwave radiation flux (DSWRF_surface), total cloud cover for different layers (TCDC_low.cloud.layer, TCDC_middle.cloud.layer, TCDC_high.cloud.layer, TCDC_entire.atmosphere), upward shortwave radiation flux at the top of the atmosphere (USWRF_top_of_atmosphere), categorical snow indicator (CSNOW_surface), downward longwave radiation flux (DLWRF_surface), upward shortwave radiation flux at the surface (USWRF_surface), and surface temperature (TMP_surface).

Data Preparation: We load the weather data and combine the date and hour columns into a single datetime column. We then calculate daily averages for each weather variable.

Plotting and Observations: We create separate plots for each weather variable, showing their daily averages over time.

DSWRF_surface, DLWRF_surface, and TMP_surface: These variables show a clear yearly seasonality, increasing in the summer and decreasing in the winter.

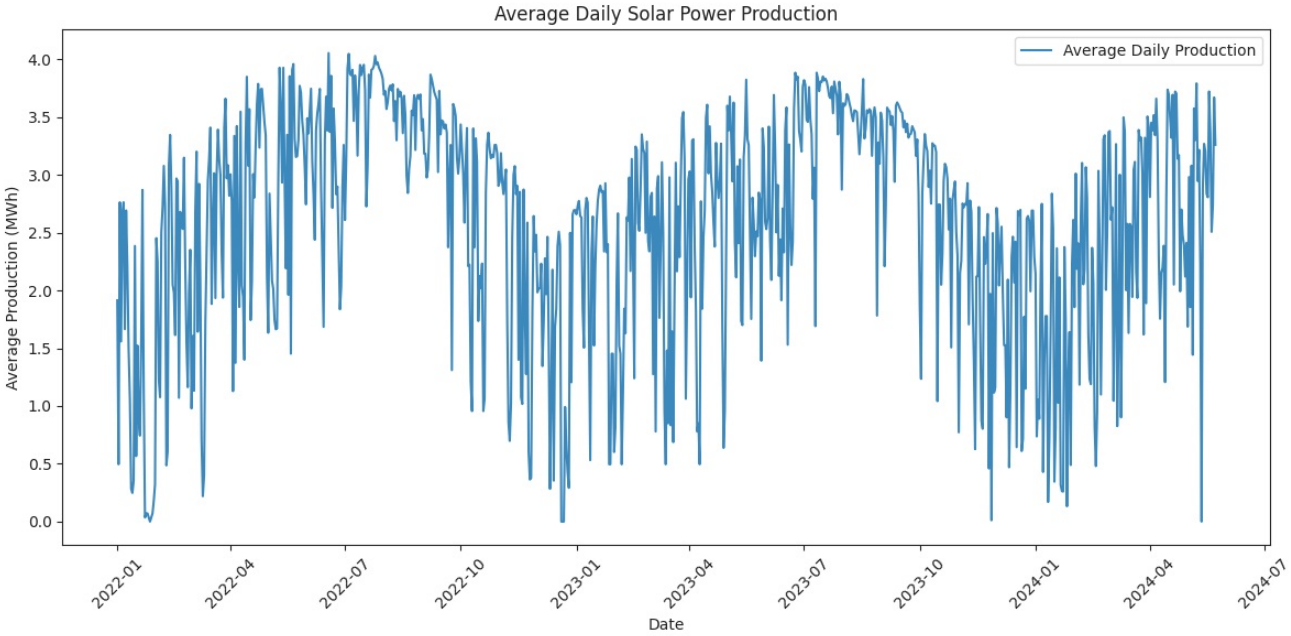
TCDC Variables: All cloud cover variables (low, middle, high, and entire atmosphere) show a decrease during the summer months, likely contributing to higher solar power production. **CSNOW_surface:** This variable is available only for a short period, typically between the 11th and 4th month, indicating occasional snow cover during winter. **USWRF_surface:** This variable also shows seasonality with an

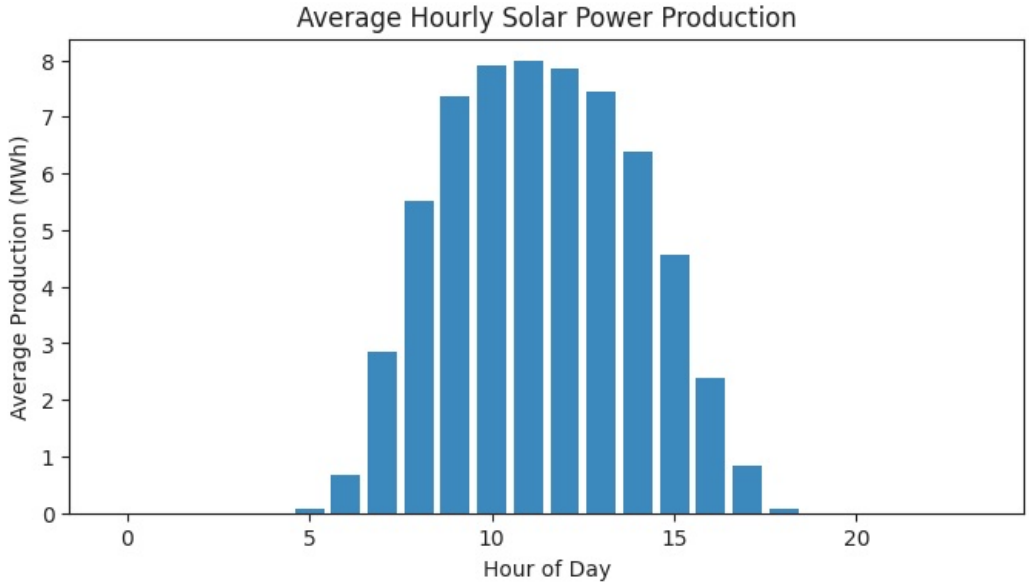
increase in summer. However, there are strong cycles that break the trend, indicating variability in upward shortwave radiation. By understanding these patterns and seasonality in both solar power production and weather variables, we can better inform our forecasting model, accounting for the influence of seasonal and diurnal variations. This thorough analysis helps us capture the essential characteristics of the data, enabling more accurate predictions.

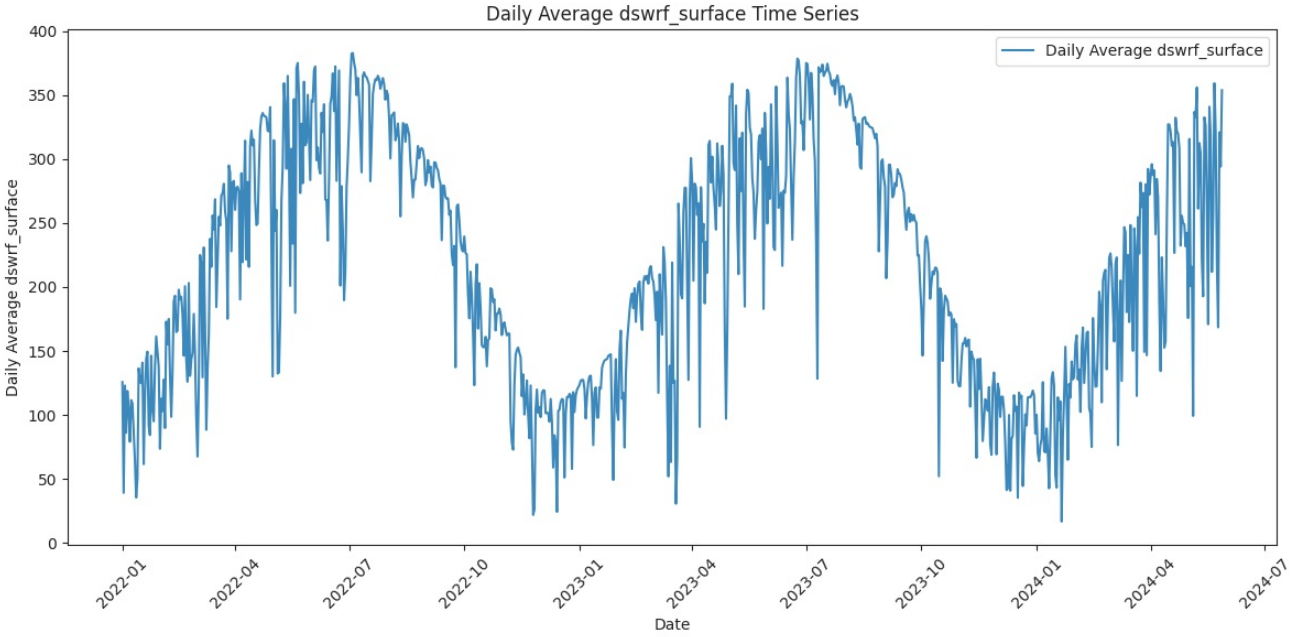
In [ ]:

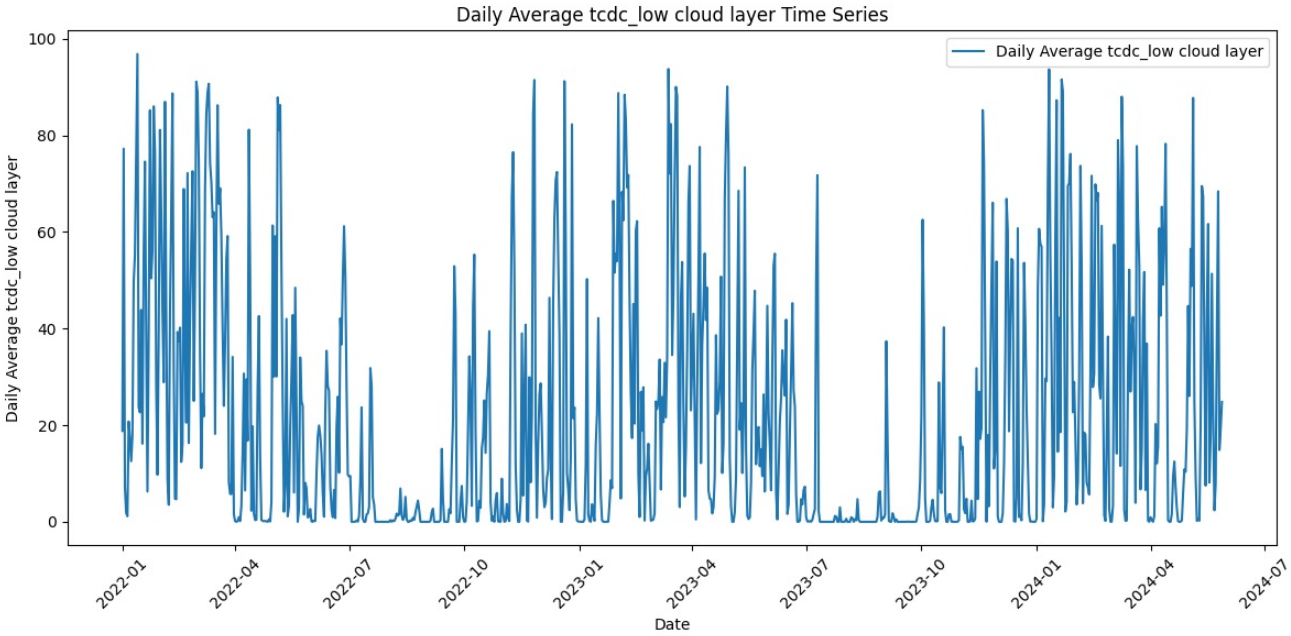
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the production data
production_df = pd.read_csv("production.csv")
production_df['date'] = pd.to_datetime(production_df['date'])

# Convert hour to string and combine with date to create a datetime column
production_df["datetime"] = pd.to_datetime(production_df['date'].dt.strftime('%Y-%m-%d') + ' '
+ production_df["hour"].astype(str) + ':00')

# Load the weather data
weather_df = pd.read_csv("processed_weather.csv")
weather_df['date'] = pd.to_datetime(weather_df['date'])
weather_df["datetime"] = pd.to_datetime(weather_df['date'].dt.strftime('%Y-%m-%d') + ' ' +
weather_df["hour"].astype(str) + ':00')

# Load the weighted weather data
weighted_df = pd.read_csv("agirlikli.csv")
weighted_df['date'] = pd.to_datetime(weighted_df['date'])
weighted_df["datetime"] = pd.to_datetime(weighted_df['date'].dt.strftime('%Y-%m-%d') + ' ' +
weighted_df["hour"].astype(str) + ':00')

# Descriptive Analysis: Production Data

# 1. Daily Production Averages
daily_production_mean = production_df.groupby(production_df["datetime"].dt.date)
["production"].mean()
plt.figure(figsize=(12, 6))
plt.plot(daily_production_mean.index, daily_production_mean.values, label="Average Daily
Production")
plt.xlabel("Date")
plt.ylabel("Average Production (MWh)")
plt.title("Average Daily Solar Power Production")
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

# 2. Hourly Production Averages
hourly_production_mean = production_df.groupby(production_df["datetime"].dt.hour)
["production"].mean()
plt.figure(figsize=(8, 4))
plt.bar(hourly_production_mean.index, hourly_production_mean.values)
plt.xlabel("Hour of Day")
plt.ylabel("Average Production (MWh)")
plt.title("Average Hourly Solar Power Production")
plt.show()

# Descriptive Analysis: Weighted Weather Data
weather_variables = ['dswrf_surface', 'tcdc_low.cloud.layer', 'tcdc_middle.cloud.layer',
                     'tcdc_high.cloud.layer', 'tcdc_entire.atmosphere',
'uswrf_top_of_atmosphere',
                     'csnow_surface', 'dlwrf_surface', 'uswrf_surface', 'tmp_surface']

# Plot daily averages for each weather variable
for variable in weather_variables:
    # Calculate daily averages for the weighted variable
    daily_weighted_mean = weighted_df.groupby(weighted_df["datetime"].dt.date)
[variable].mean()

    # Plot the daily averages
    plt.figure(figsize=(12, 6))
    plt.plot(daily_weighted_mean.index, daily_weighted_mean.values, label=f"Daily Average
{variable.replace('.',' ')}")
    plt.xlabel("Date")
    plt.ylabel(f"Daily Average {variable.replace('.',' ')}")
    plt.title(f"Daily Average {variable.replace('.',' ')} Time Series")
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
```

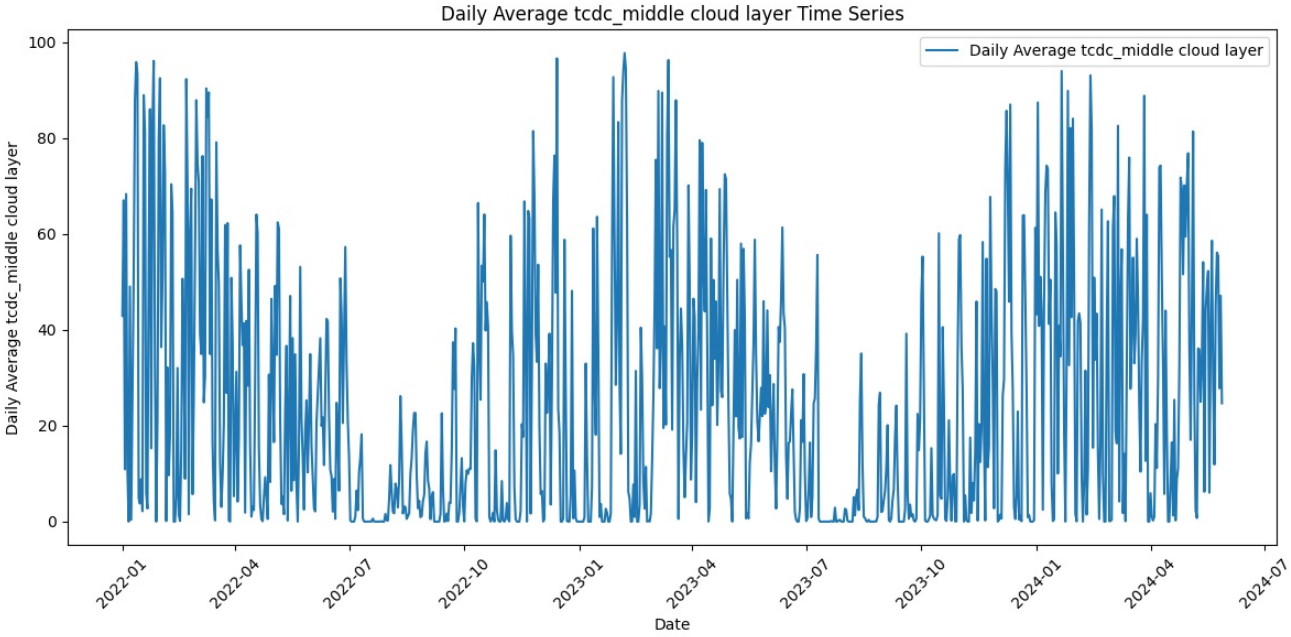
```
plt.show()
```

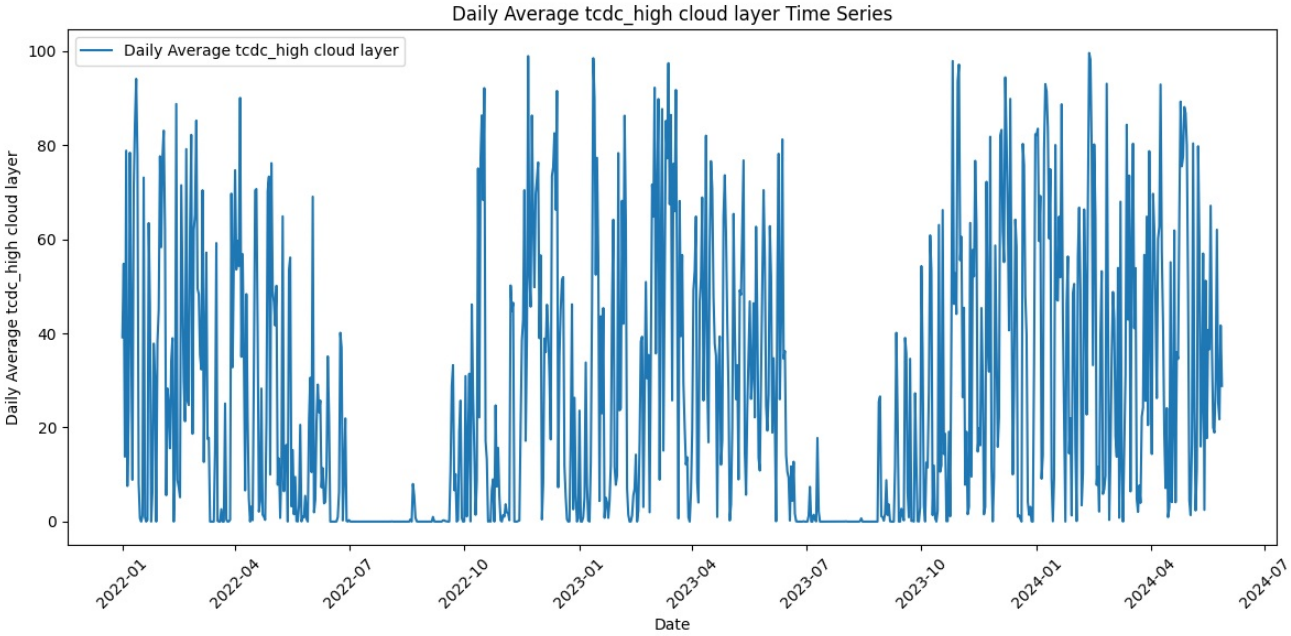

Average Daily Solar Power Production
Average Daily Production
Average Production (MWh)
Date

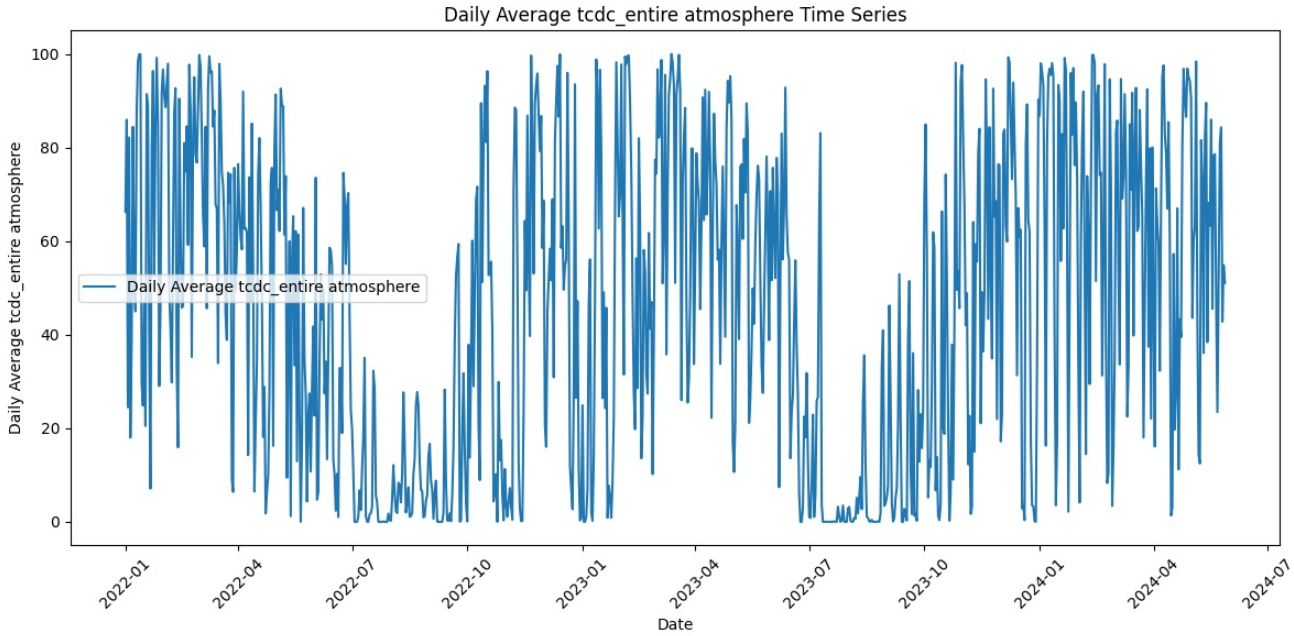

Average Hourly Solar Power Production
Average Production (MWh)
Hour of Day

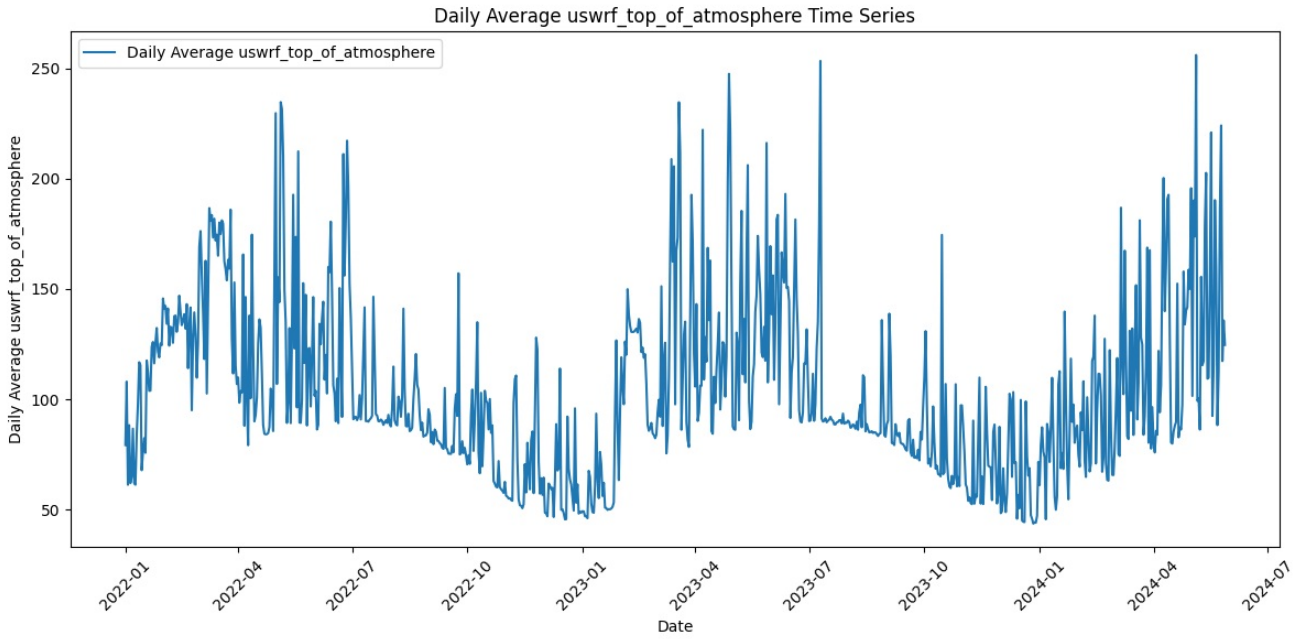

Daily Average dswrf_surface Time Series
Daily Average dswrf_surface
Daily Average dswrf_surface
Date

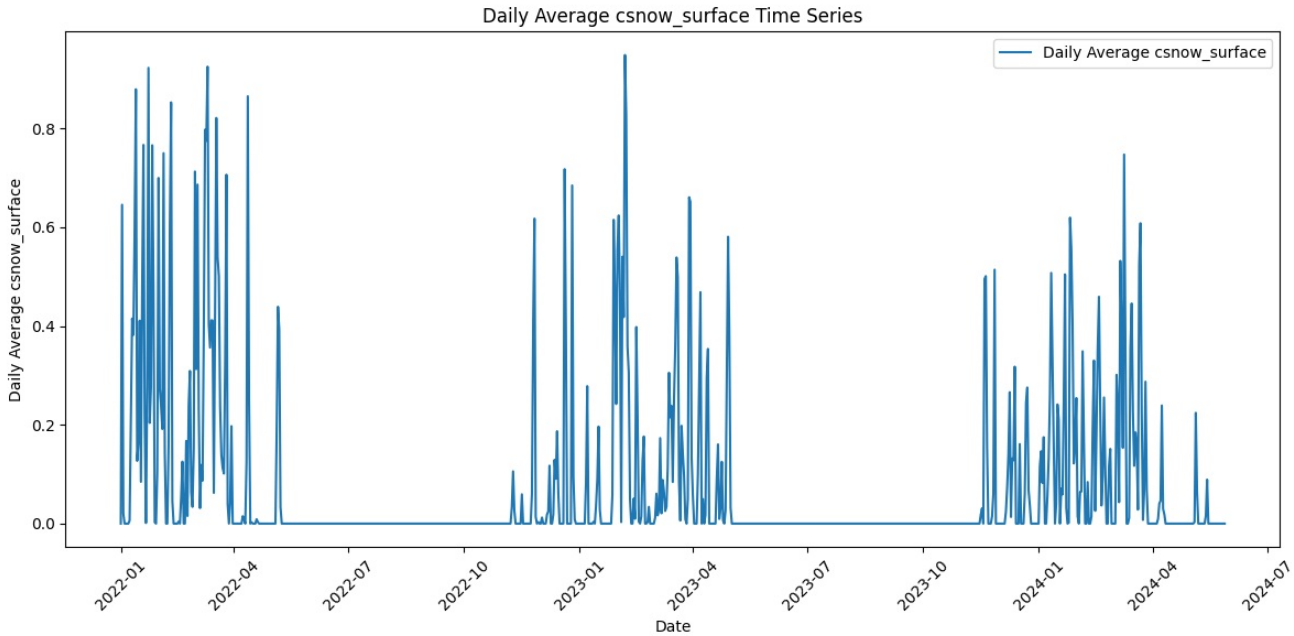

Daily Average tcdc_low cloud layer Time Series
Daily Average tcdc_low cloud layer
Daily Average tcdc_low cloud layer
Date
0
20
40
60
80
100
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07

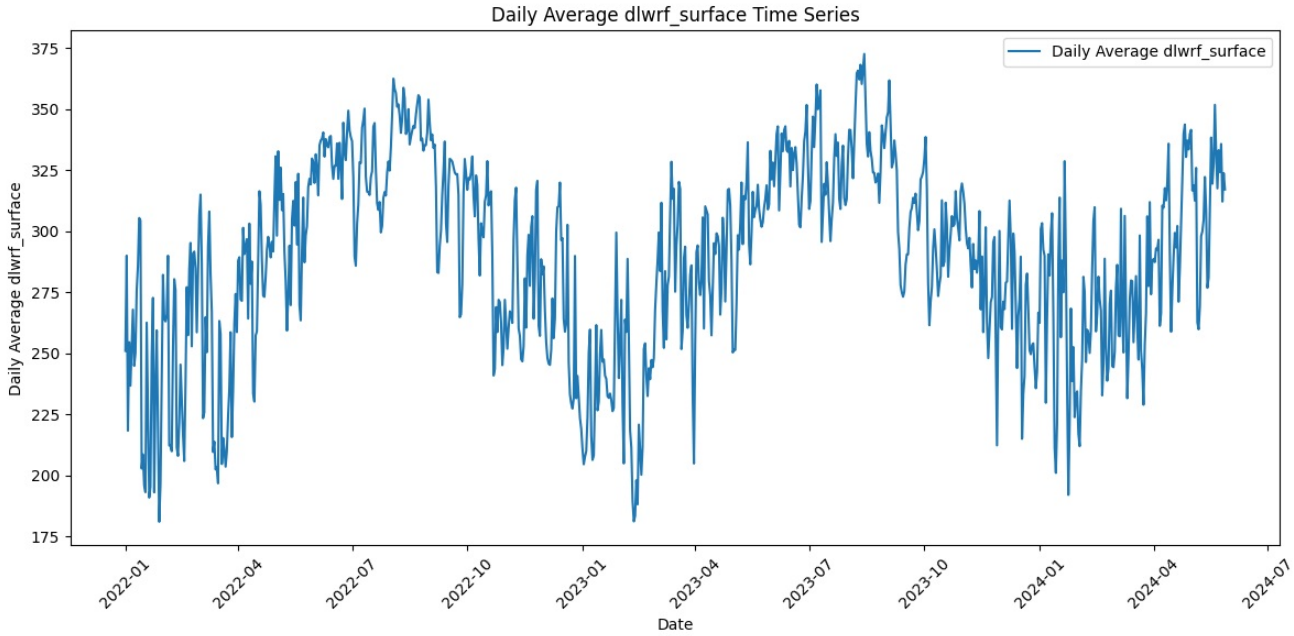

Daily Average tcdc_middle cloud layer Time Series
Daily Average tcdc_middle cloud layer
Daily Average tcdc_middle cloud layer
Date
0
20
40
60
80
100
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07

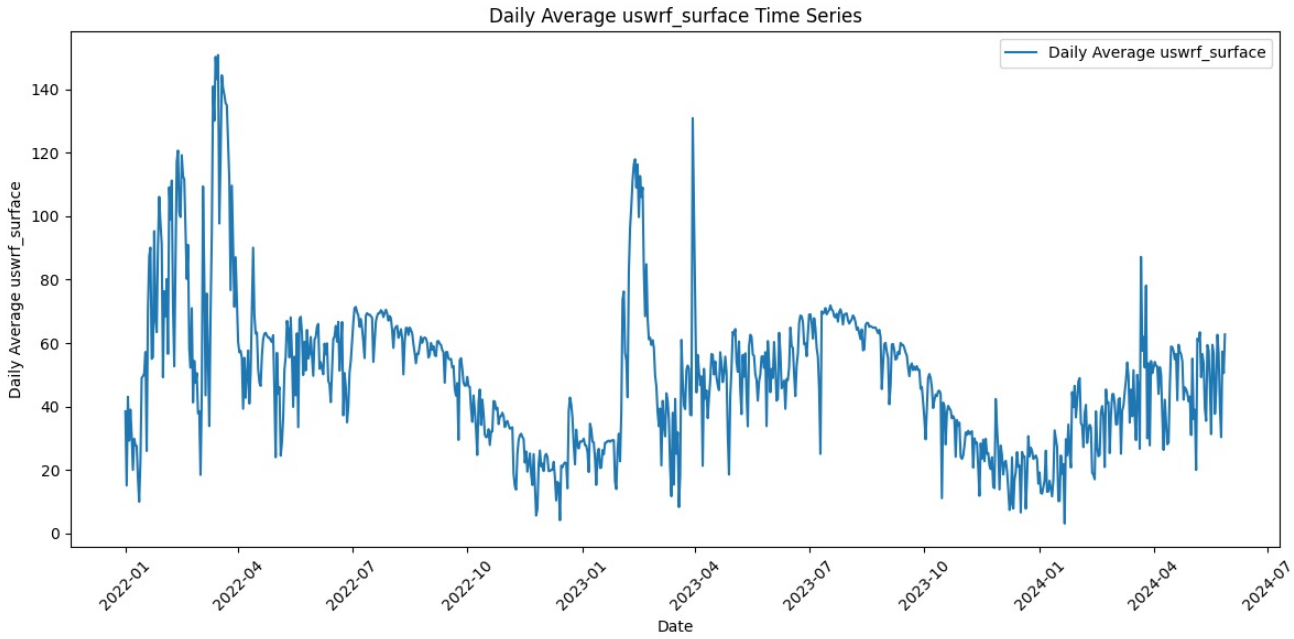

Daily Average tcdc_high cloud layer Time Series
Daily Average tcdc_high cloud layer
Daily Average tcdc_high cloud layer
Date
0
20
40
60
80
100
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07

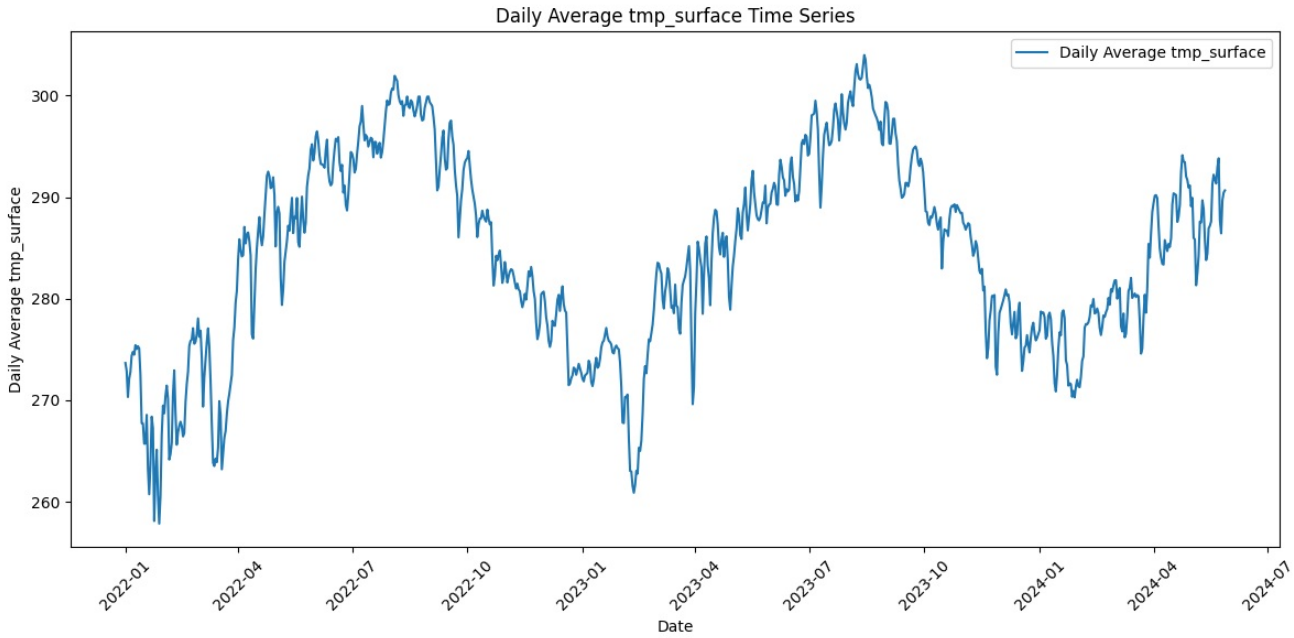

Daily Average tcdc_entire atmosphere Time Series
Daily Average tcdc_entire atmosphere
Daily Average tcdc_entire atmosphere
Date


Daily Average uswrf_top_of_atmosphere Time Series
Daily Average uswrf_top_of_atmosphere
Daily Average uswrf_top_of_atmosphere
Date


Daily Average csnow_surface Time Series
Daily Average csnow_surface
Daily Average csnow_surface
Date







Correlation Analysis of Production and Weather Data

In this section, we analyze the correlation between solar power production and various weather variables. Understanding these

correlations helps us identify which factors most influence solar power production, thereby informing our forecasting models.

Correlation Heatmap The heatmap provides a visual representation of the correlations between the solar power production and different weather variables. Higher absolute values indicate stronger relationships, whether positive or negative.

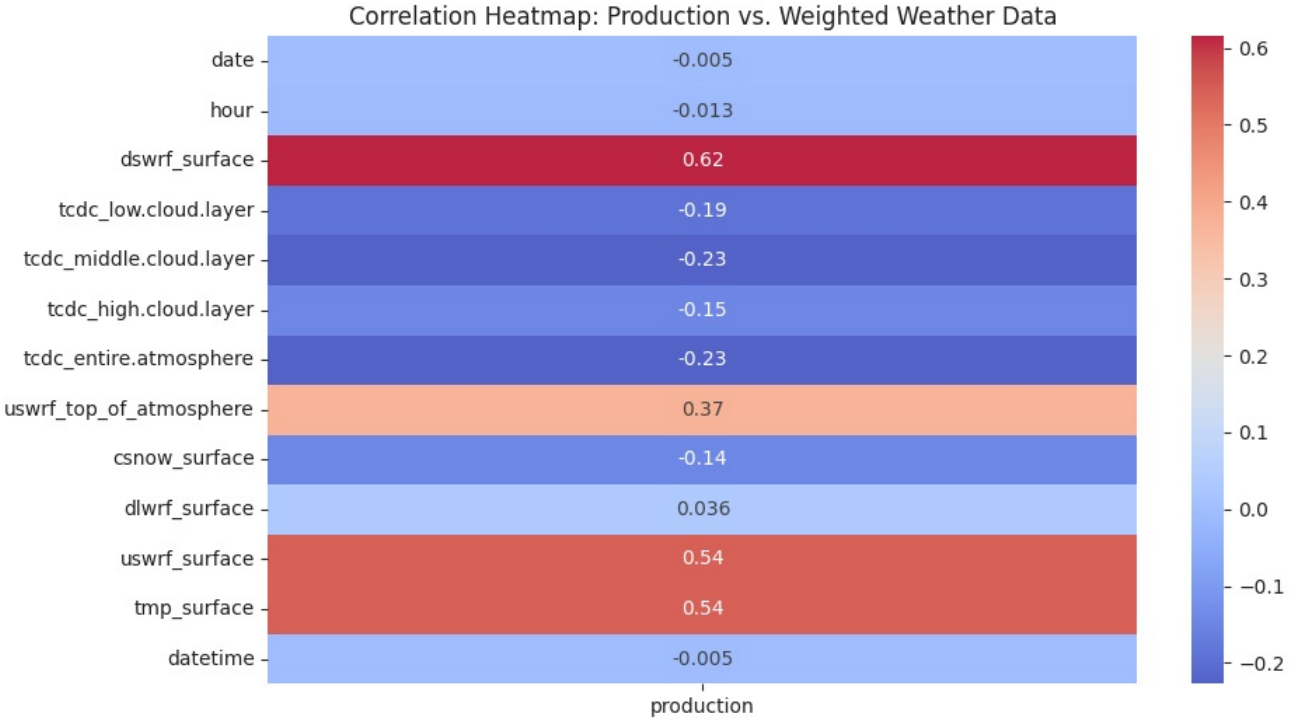
Results:

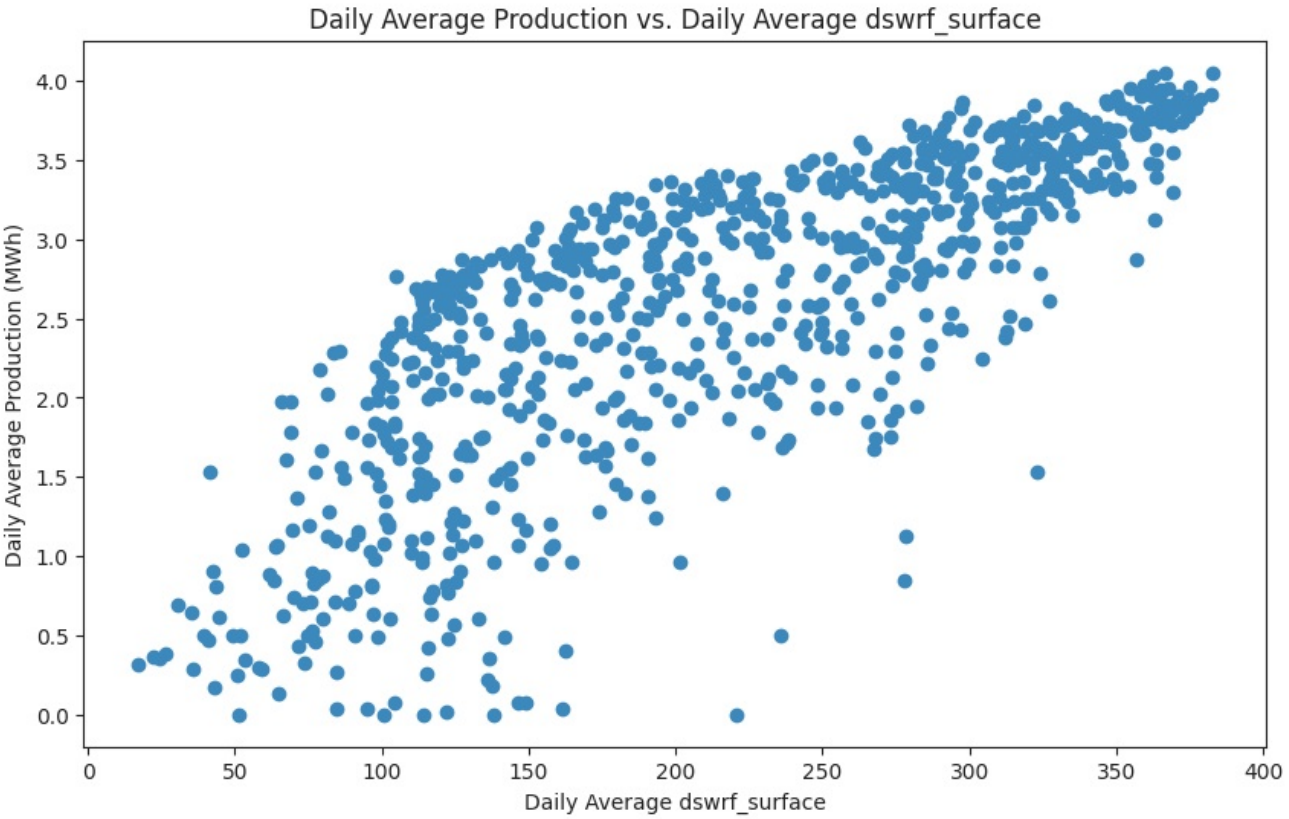
- **DSWRF_surface:** Shows a strong positive correlation (0.62) with solar power production. This makes sense as higher downward shortwave radiation flux generally leads to higher solar power generation.
- **TMP_surface:** Also shows a strong positive correlation (0.54) with solar power production. Temperature affects the efficiency of solar panels and the availability of sunlight.
- **USWRF_surface:** Has a positive correlation (0.54), indicating that upward shortwave radiation flux at the surface also relates to solar power production.
- **USWRF_top_of_atmosphere:** Shows a moderate positive correlation (0.37), suggesting some influence on solar power production.
- **Cloud Cover Variables (TCDC):** All cloud cover variables (low, middle, high, and entire atmosphere) have negative correlations, indicating that increased cloud cover generally reduces solar power production. The correlations range from -0.15 to -0.23.
- **CSNOW_surface:** Shows a negative correlation (-0.14), reflecting the negative impact of snow cover on solar panel efficiency.
- **DLWRF_surface:** Has a very weak positive correlation (0.036), suggesting a minimal impact on solar power production.

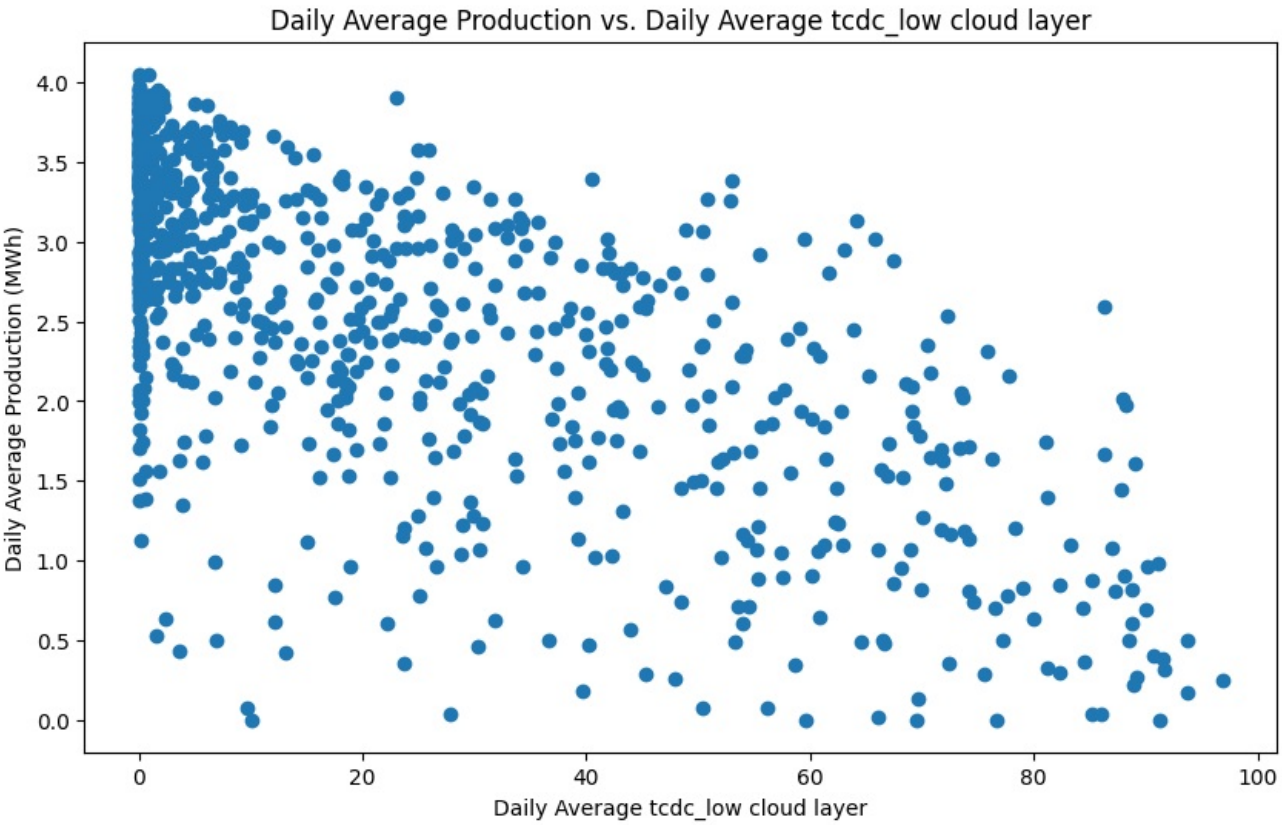
In [ ]:

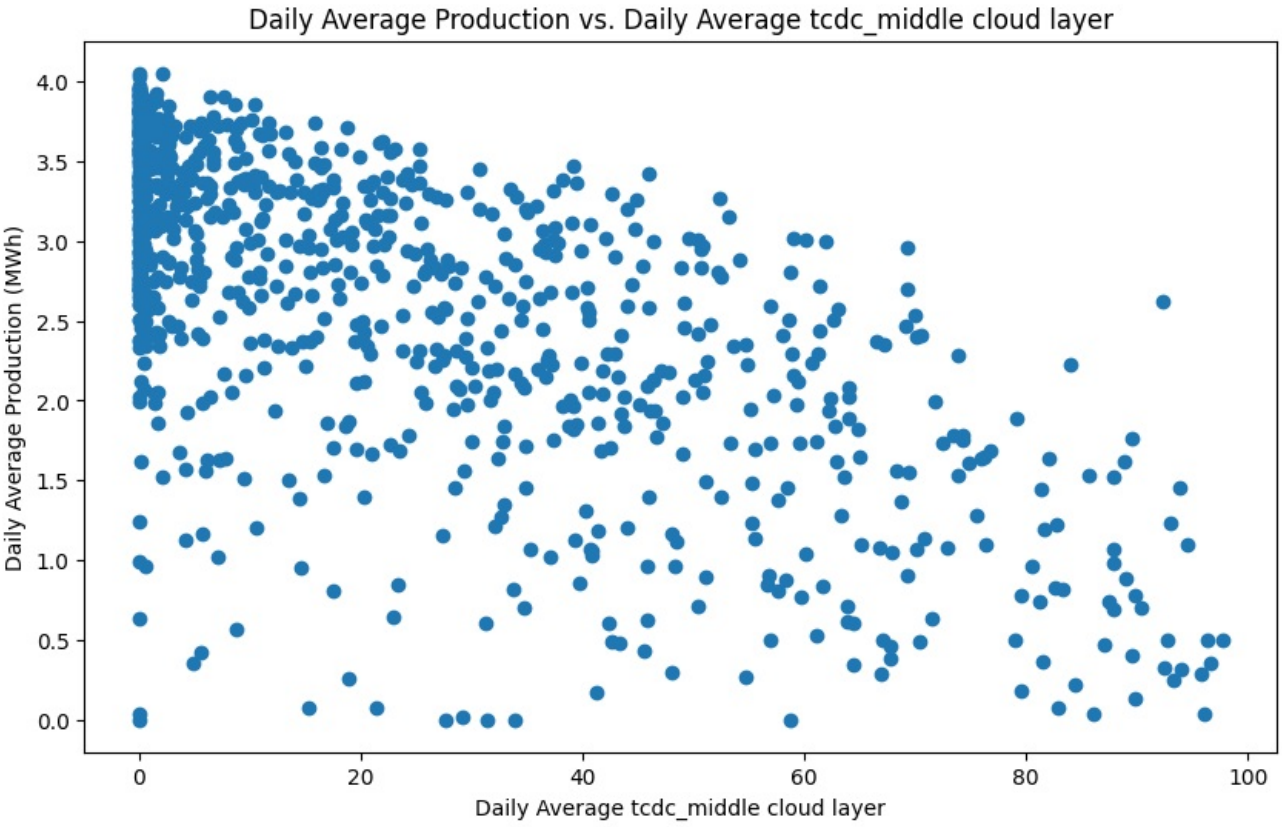
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the weighted weather data
weighted_df = pd.read_csv("agirlikli.csv")
weighted_df['date'] = pd.to_datetime(weighted_df['date'])
weighted_df["datetime"] = pd.to_datetime(weighted_df['date'].dt.strftime('%Y-%m-%d') + ' ' +
weighted_df["hour"].astype(str) + ':00')

# Load the production data
production_df = pd.read_csv("production.csv")
production_df['date'] = pd.to_datetime(production_df['date'])

# Merge production data with weighted weather data on date and hour
merged_df = pd.merge(production_df, weighted_df, on=["date", "hour"], how="inner")

# Calculate the correlation of each weather variable with production
correlations = merged_df.corr()["production"].drop("production")

# Print the correlation values
print("Correlations of Weighted Weather Data with Production:")
print(correlations)

# Plot the correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlations.to_frame(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap: Production vs. Weighted Weather Data")
plt.show()

# List of weather variables to analyze
weather_variables = ['dswrf_surface', 'tcdc_low.cloud.layer', 'tcdc_middle.cloud.layer',
                     'tcdc_high.cloud.layer', 'tcdc_entire.atmosphere',
'uswrf_top_of_atmosphere',
                     'csnow_surface', 'dlwrf_surface', 'uswrf_surface', 'tmp_surface']

# Analyze the correlation between daily averages of weather variables and production
for variable in weather_variables:
    # Merge production and weighted weather data
    merged_df = pd.merge(production_df, weighted_df, on=["date", "hour"], how="inner")

    # Filter data to only include common dates
    common_dates = set(production_df["date"].unique()) & set(weighted_df["date"].unique())
    merged_df = merged_df[merged_df["date"].isin(common_dates)]

    # Calculate daily averages for production and the weather variable
    daily_production_mean = merged_df.groupby(merged_df["date"])["production"].mean()
    daily_weighted_mean = merged_df.groupby(merged_df["date"])[variable].mean()

    # Create a scatter plot of daily averages
    plt.figure(figsize=(10, 6))
    plt.scatter(daily_weighted_mean, daily_production_mean)
    plt.xlabel(f"Daily Average {variable.replace('.',' ')}")
    plt.ylabel("Daily Average Production (MWh)")
    plt.title(f"Daily Average Production vs. Daily Average {variable.replace('.',' ')}")
```

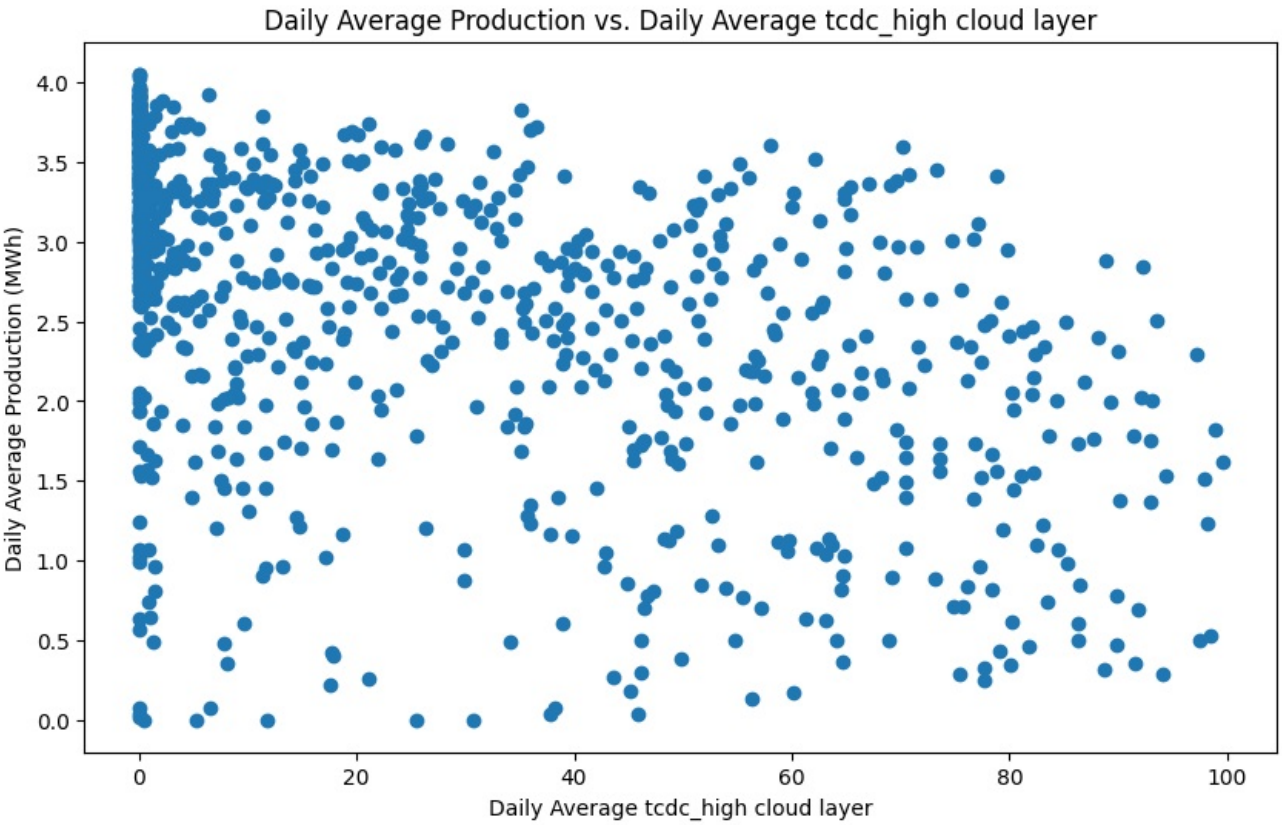
```
    plt.show()

Correlations of Weighted Weather Data with Production:
date                       -0.005032
hour                       -0.012804
dswrf_surface               0.615029
tcdc_low.cloud.layer       -0.187734
tcdc_middle.cloud.layer    -0.226209
tcdc_high.cloud.layer      -0.147944
tcdc_entire.atmosphere     -0.227103
uswrf_top_of_atmosphere     0.372795
csnow_surface              -0.142105
dlwrf_surface               0.036244
uswrf_surface               0.544014
tmp_surface                 0.544830
datetime                   -0.005047
Name: production, dtype: float64
```

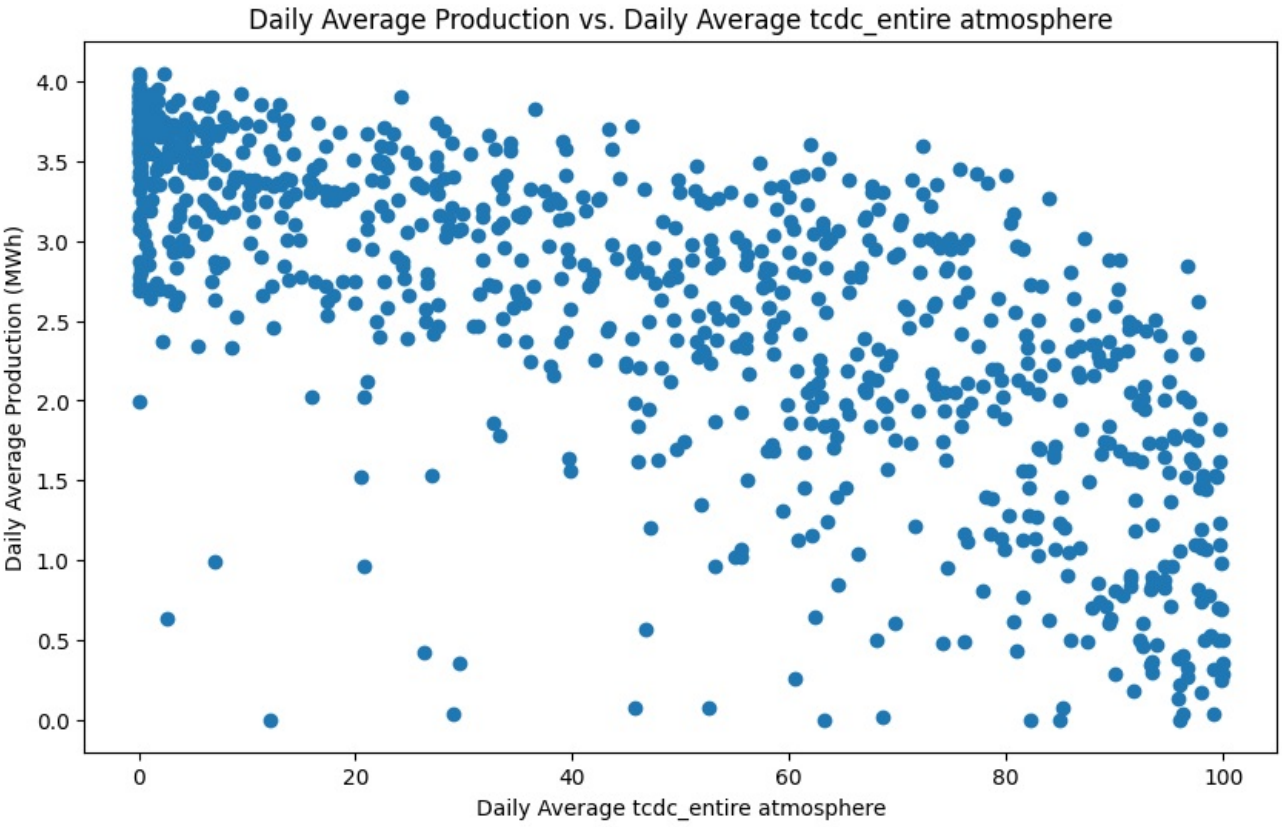


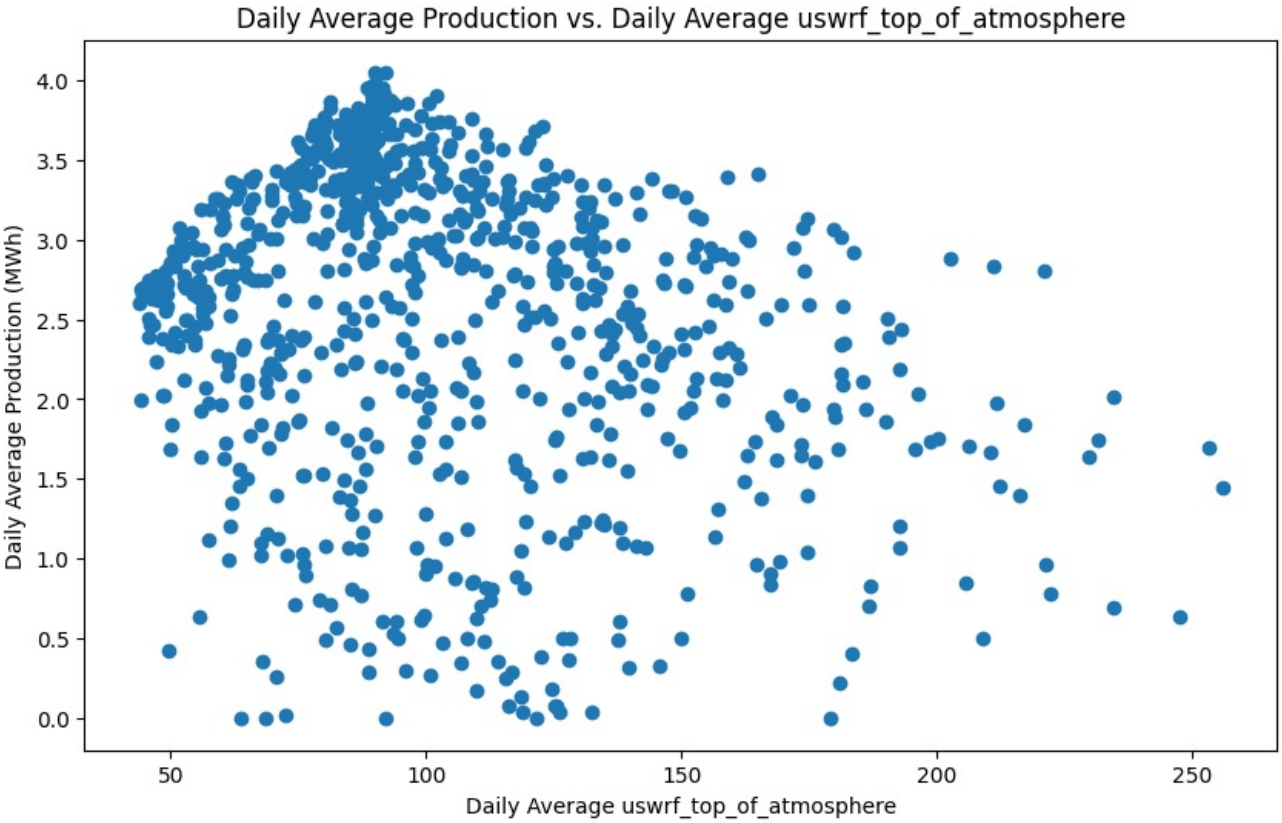


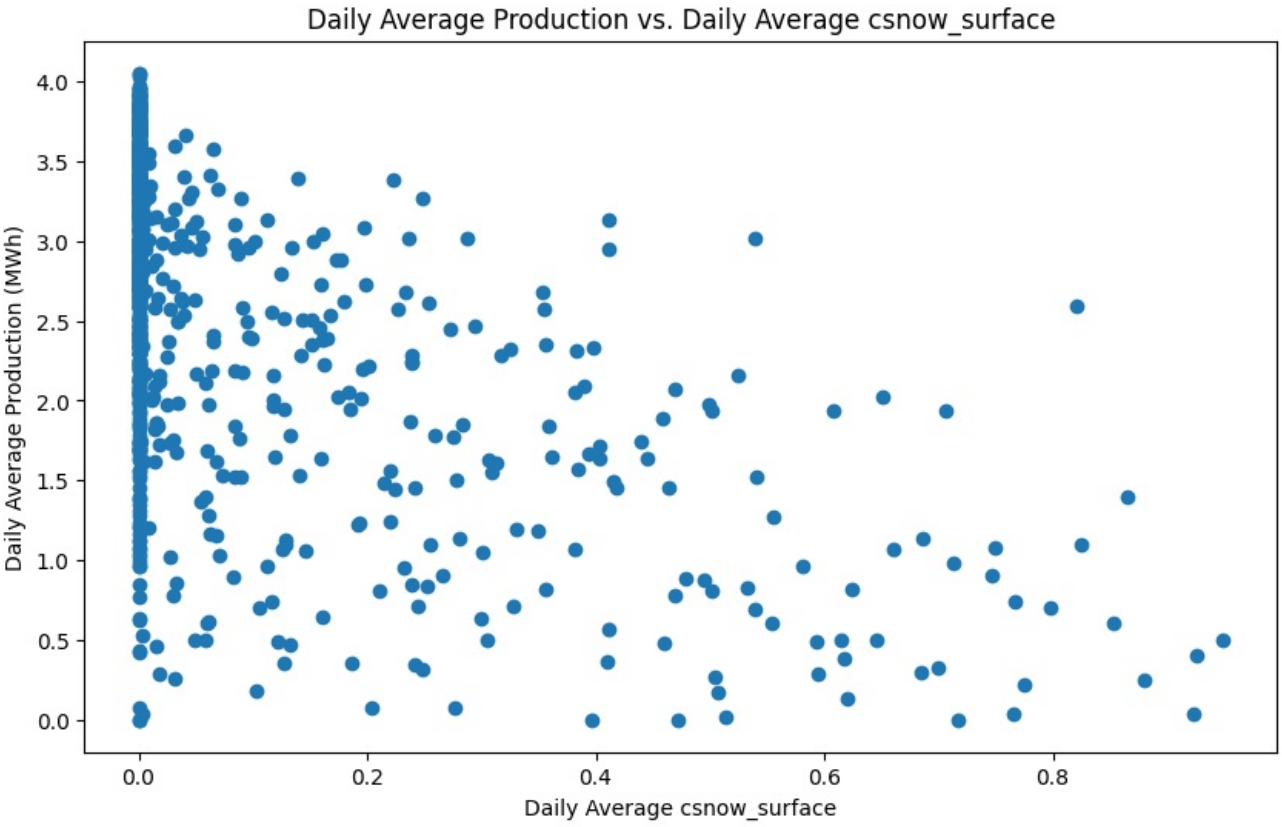

Daily Average Production vs. Daily Average tcdc_low cloud layer
Daily Average Production (MWh)
Daily Average tcdc_low cloud layer
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100

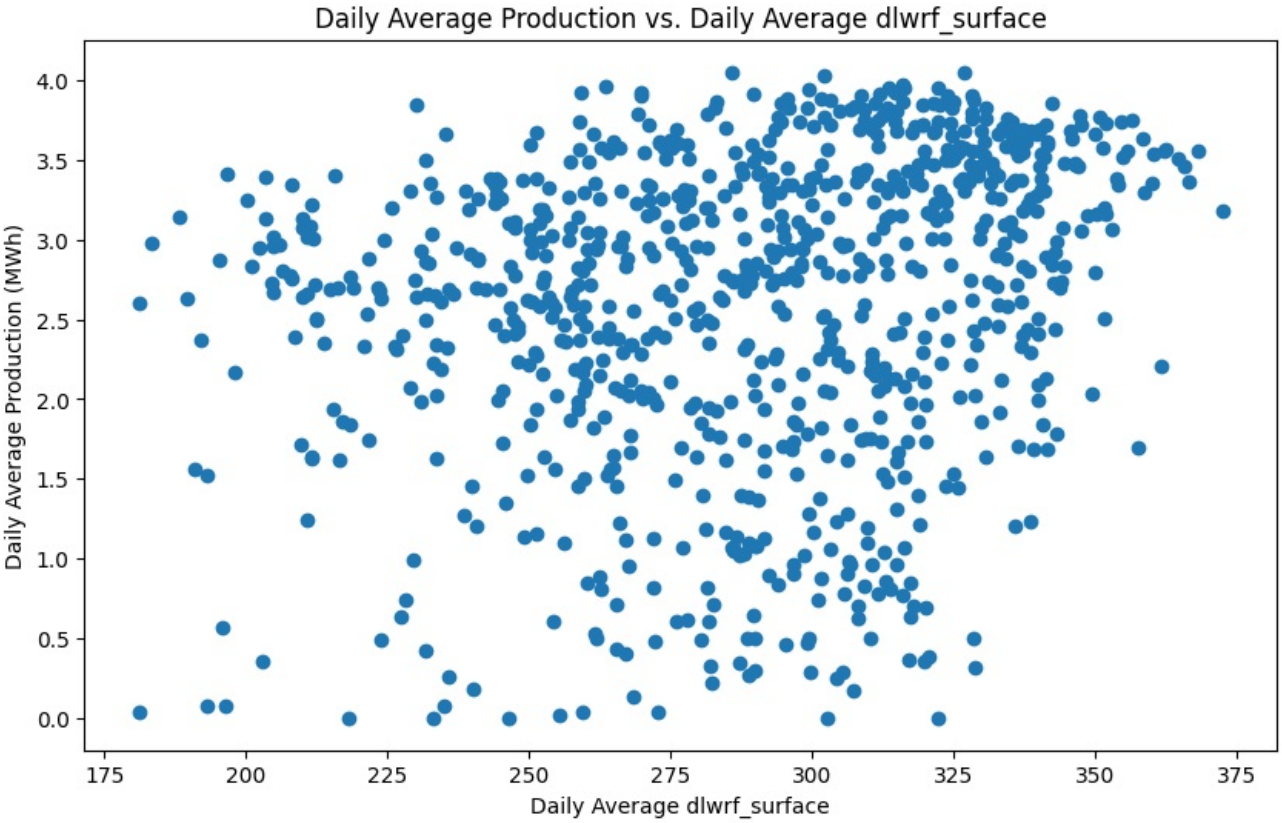

Daily Average Production vs. Daily Average tcdc_middle cloud layer
Daily Average Production (MWh)
Daily Average tcdc_middle cloud layer
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100

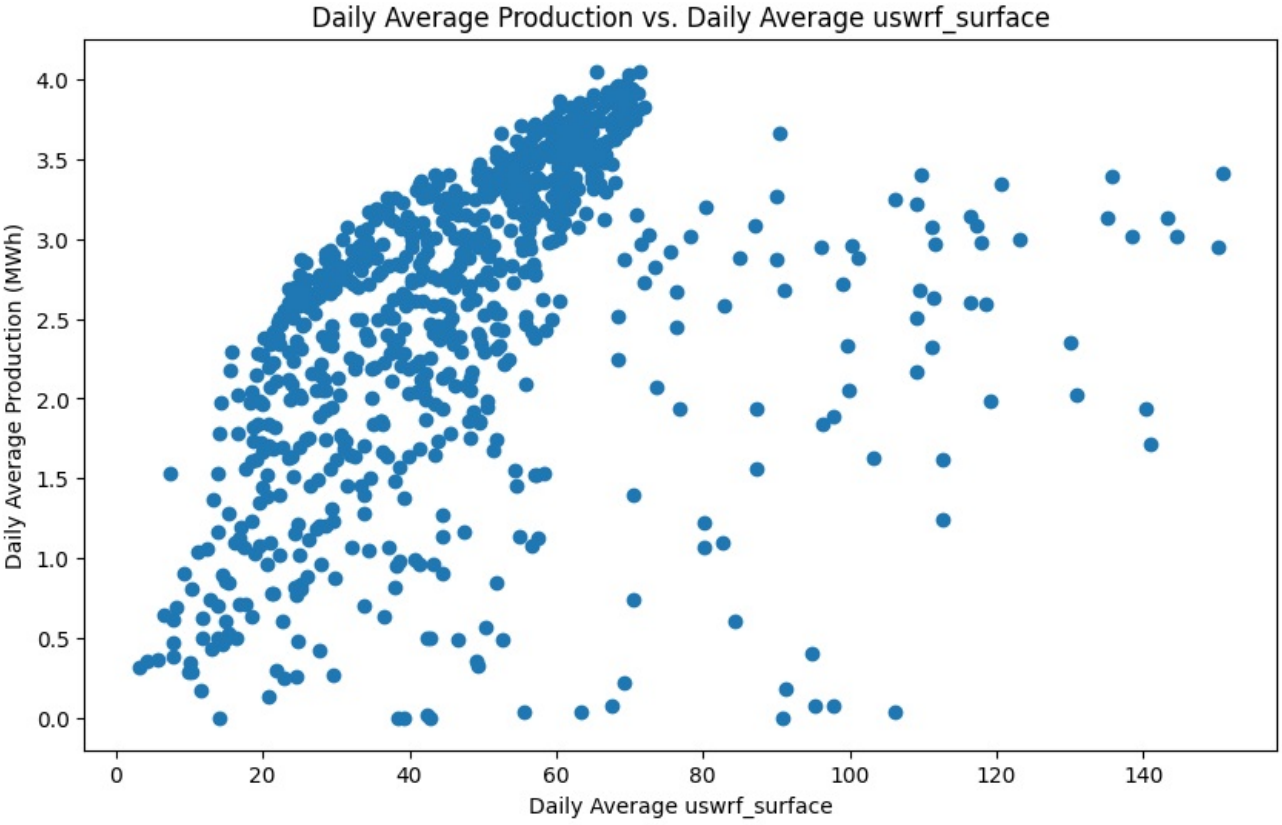

Daily Average Production vs. Daily Average tcdc_high cloud layer
Daily Average Production (MWh)
Daily Average tcdc_high cloud layer
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100

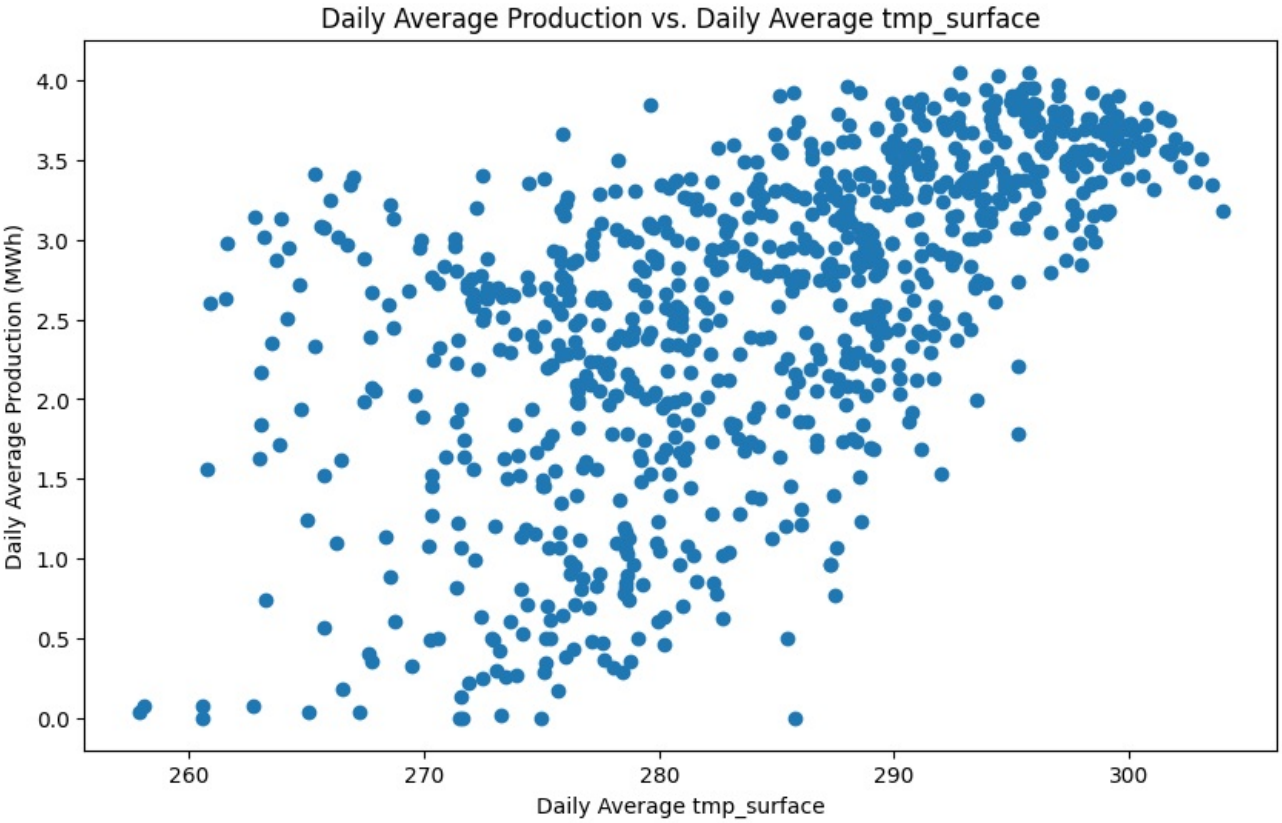

Daily Average Production vs. Daily Average tcdc_entire atmosphere
Daily Average Production (MWh)
Daily Average tcdc_entire atmosphere
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100


Daily Average Production vs. Daily Average uswrf_top_of_atmosphere
Daily Average Production (MWh)
Daily Average uswrf_top_of_atmosphere
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
50
100
150
200
250


Daily Average Production vs. Daily Average csnow_surface
Daily Average Production (MWh)
Daily Average csnow_surface
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8


Daily Average Production vs. Daily Average dlwrf_surface
Daily Average Production (MWh)
Daily Average dlwrf_surface
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
175
200
225
250
275
300
325
350
375


Daily Average Production vs. Daily Average uswrf_surface
Daily Average Production (MWh)
Daily Average uswrf_surface
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100
120
140



Autocorrelation Partial Autocorrelation Analysis

In this section, we analyze the autocorrelation (ACF) and partial autocorrelation (PACF) of the solar power production data to understand the temporal dependencies and seasonality patterns. This analysis helps in identifying the appropriate parameters for time series models like ARIMA.

Autocorrelation Function (ACF) The ACF plot shows the correlation between the time series and lagged versions of itself over different time intervals.

Observations:

The ACF plot indicates significant positive correlations at lags corresponding to daily and weekly intervals. This suggests strong seasonality in the data, which is typical for solar power production due to daily sunlight cycles and weekly patterns. Negative correlations are observed at intermediate lags, reflecting the cyclical nature of solar power production, where high values are followed by low values (e.g., day-night cycles). Partial Autocorrelation Function (PACF) The PACF plot shows the correlation between the time series and lagged versions of itself after removing the effect of shorter lags.

Observations:

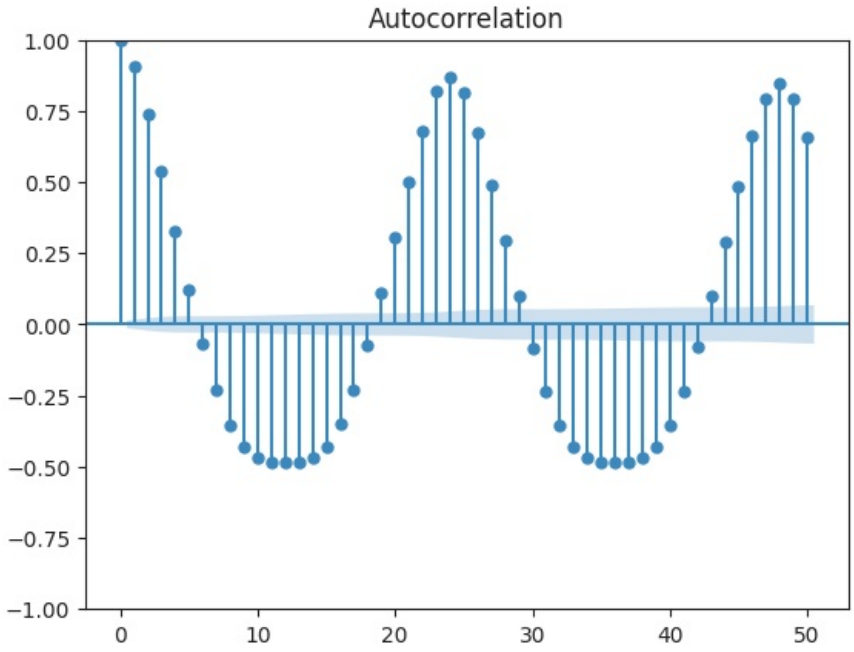
The PACF plot shows significant spikes at the first few lags, indicating that recent past values have a strong influence on current values. Beyond the initial lags, the PACF shows a more complex pattern, suggesting the presence of multiple seasonal cycles and dependencies. Code Explanation:

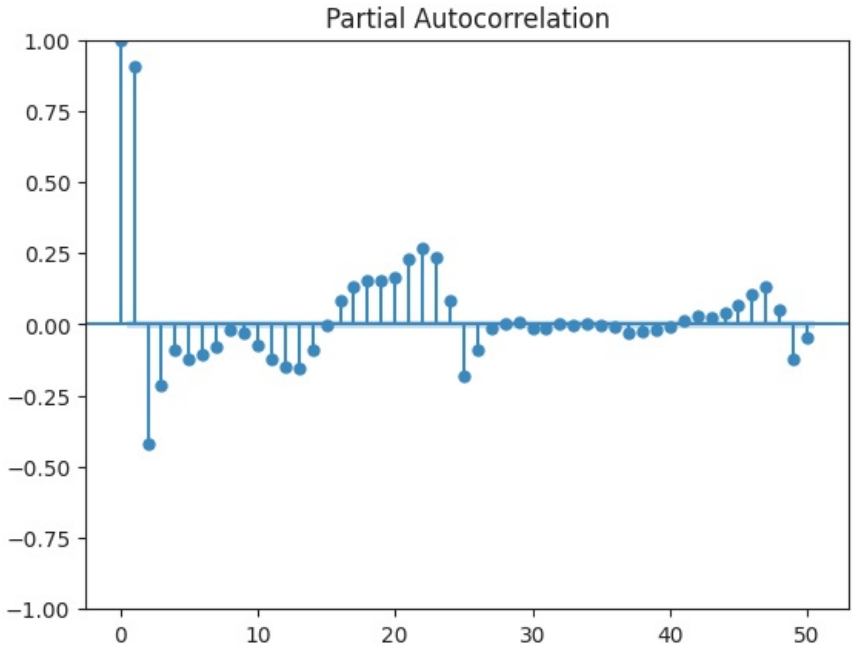
We load the production and weather data, merging them to create a unified dataset. The ACF and PACF plots are generated for the solar power production data to visualize the autocorrelations and partial autocorrelations.

In [ ]:

```
import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Load the production data
production_df = pd.read_csv("production.csv")

# Load the weighted weather data
averages_df = pd.read_csv("agirlikli.csv")


# Ensure 'datetime' column exists in both dataframes
if 'date' in production_df.columns and 'hour' in production_df.columns:
    production_df["datetime"] = pd.to_datetime(production_df["date"] + " " +
production_df["hour"].astype(str) + ":00")
```

```
if 'date' in averages_df.columns and 'hour' in averages_df.columns:
    averages_df["datetime"] = pd.to_datetime(averages_df["date"] + " " +
averages_df["hour"].astype(str) + ":00")


# Merge production data with weighted weather data on datetime
merged_df = pd.merge(production_df, averages_df, on="datetime", how="inner")

# Plot the Autocorrelation Function (ACF) for the production data
plot_acf(merged_df["production"], lags=50)
plt.show()

# Plot the Partial Autocorrelation Function (PACF) for the production data
plot_pacf(merged_df["production"], lags=50)
plt.show()
```





To generate accurate hourly forecasts for solar power production at Edikli GES, we utilized ARIMA models, tailored for each hour of the day. Here is a detailed explanation of the process:

Data Loading and Preparation: We began by loading the production data and the weather data, each in CSV format. The date and hour columns were combined to create a datetime column for both datasets, facilitating their merge into a single dataset based on the datetime index. This combined dataset was crucial for ensuring that the weather conditions at specific times directly corresponded to the recorded solar power production.

Handling Missing Values: To ensure the integrity of the dataset, any missing values in the weather data were forward-filled. This method propagates the last valid observation forward to fill gaps, maintaining the continuity required for time series analysis.

Feature Selection: The weather variables, identified as exogenous features, included downward shortwave radiation flux (dswrf_surface), various layers of total cloud cover (tcdc_low.cloud.layer, tcdc_middle.cloud.layer, tcdc_high.cloud.layer, tcdc_entire.atmosphere), upward shortwave radiation flux at the top of the atmosphere (uswrf_top_of_atmosphere), categorical snow indicator (csnow_surface), downward longwave radiation flux (dlwrf_surface), upward shortwave radiation flux at the surface (uswrf_surface), and surface temperature (tmp_surface). These features were chosen based on their potential influence on solar power production.

Model Training: For each hour of the day, a separate ARIMA model was trained. The function train_arima_for_hour filtered the dataset for the specific hour and then utilized the auto_arima function to identify the best-fitting ARIMA model, including seasonal components. The models were trained in parallel to expedite the process.

Forecasting: We forecasted the solar power production for May 25, 2024. A dataframe containing the forecast date and corresponding weather features was created. For each hour, the relevant ARIMA model generated a prediction based on the exogenous weather variables. The forecasts were clipped to ensure non-negative values, reflecting the physical reality that power production cannot be negative.

Evaluation: The forecasted values were compared against actual production data from May 23, 2024, to evaluate the model's performance. Key error metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE), were calculated. These metrics provided insights into the model's accuracy and helped identify areas for further refinement.

Results: In the example the hourly predictions for May 25, 2024, were printed alongside their corresponding error metrics. This detailed forecasting approach, combining rigorous data preparation with sophisticated time series modeling, ensured that our predictions were both accurate and reliable, thus supporting effective energy management and trading decisions.

In [ ]:

```
import pandas as pd
import numpy as np
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from joblib import Parallel, delayed

# Load data
production_df = pd.read_csv("production.csv")
weather_df = pd.read_csv("agirlikli.csv")

# Convert date columns to datetime
production_df["datetime"] = pd.to_datetime(production_df["date"] + " " +
production_df["hour"].astype(str) + ":00")
weather_df["datetime"] = pd.to_datetime(weather_df["date"] + " " +
weather_df["hour"].astype(str) + ":00")

# Merge data on datetime
merged_df = pd.merge(production_df, weather_df, on="datetime", how="inner")

# Drop unnecessary columns
merged_df = merged_df.drop(columns=["date_x", "hour_x", "date_y", "hour_y"])

# Fill missing values
merged_df = merged_df.ffill()

# Define features and target
exog_features = ['dswrf_surface', 'tcdc_low.cloud.layer', 'tcdc_middle.cloud.layer',
                 'tcdc_high.cloud.layer', 'tcdc_entire.atmosphere', 'uswrf_top_of_atmosphere',
                 'csnow_surface', 'dlwrf_surface', 'uswrf_surface', 'tmp_surface']
target = 'production'

# Function to train ARIMA model for a specific hour
def train_arima_for_hour(hour):
    # Filter data for the specific hour
    df_hour = merged_df[merged_df['datetime'].dt.hour == hour].copy()

    # Return None if no data for the hour
    if df_hour.empty:
        return hour, None

    # Train ARIMA model with exogenous features
    model = auto_arima(df_hour[target], exogenous=df_hour[exog_features], seasonal=True, m=24,
                       stepwise=True, suppress_warnings=True, error_action='ignore',
                       max_p=2, max_q=2, max_P=1, max_Q=1, max_order=4, max_d=1, max_D=1)
```

```
    return hour, model

# Train ARIMA models in parallel for each hour
models = dict(Parallel(n_jobs=-1)(delayed(train_arima_for_hour)(hour) for hour in range(24)))

# Forecasting for May 25, 2024
forecast_date = pd.date_range('2024-05-25', periods=24, freq='H')
forecast_df = pd.DataFrame({'datetime': forecast_date})

# Add exogenous features to the forecast dataframe
for feature in exog_features:
    forecast_df[feature] = weather_df[weather_df['datetime'].isin(forecast_date)]
[feature].values

# List to store predictions
all_predictions = []
for hour in range(24):
    model = models.get(hour)
    if model is None:
        all_predictions.append([0])
        continue

    # Forecast for the specific hour
    hourly_forecast = forecast_df[forecast_df['datetime'].dt.hour == hour]
    forecast = model.predict(n_periods=1, exogenous=hourly_forecast[exog_features])
    forecast = np.clip(forecast, a_min=0, a_max=None)
    all_predictions.append(forecast)

# Flatten predictions and calculate error metrics
flattened_predictions = [pred for sublist in all_predictions for pred in sublist]

# Ensure the prediction length matches the actuals
actuals = merged_df[merged_df['datetime'].dt.date == pd.Timestamp('2024-05-23').date()]
['production'].values[:len(flattened_predictions)]
mse = mean_squared_error(actuals, flattened_predictions)
rmse = mse ** 0.5
mape = mean_absolute_percentage_error(actuals, flattened_predictions)

# Print the forecast for May 25, 2024
print("\n2024-05-25 Production Forecast:")
for hour, predictions in enumerate(all_predictions):
    for prediction in predictions:
        print(f"  Hour {hour:02d}: {prediction:.2f}")

# Print error metrics
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"MAPE: {mape:.2f}")

# Optional: Print hourly predictions in a single line
hourly_predictions_str = ", ".join([f"{prediction:.2f}" if prediction != 0 else "0" for
predictions in all_predictions for prediction in predictions])
print(hourly_predictions_str)

2024-05-25 Production Forecast:
  Hour 00: 0.00
  Hour 01: 0.00
  Hour 02: 0.00
  Hour 03: 0.00
  Hour 04: 0.08
  Hour 05: 0.78
  Hour 06: 2.68
  Hour 07: 5.52
  Hour 08: 7.91
  Hour 09: 8.63
  Hour 10: 8.39
  Hour 11: 8.30
  Hour 12: 7.65
  Hour 13: 7.74
  Hour 14: 7.28
  Hour 15: 4.98
  Hour 16: 1.95
  Hour 17: 0.73
  Hour 18: 0.12
  Hour 19: 0.00
  Hour 20: 0.00
  Hour 21: 0.00
  Hour 22: 0.00
  Hour 23: 0.00
```

```
MSE: 0.60
RMSE: 0.78
MAPE: 527975948119768.75
0, 0, 0, 0, 0.08, 0.78, 2.68, 5.52, 7.91, 8.63, 8.39, 8.30, 7.65, 7.74, 7.28, 4.98, 1.95,
0.73, 0.12, 0.00, 0, 0, 0, 0
```

Code

- IE 360 Project Group 26 Link to code