The goal of this notebook is to perform a thorough analysis of training strategies and model assessment through cross-validation and train-val-test splitting, and boostrapping and bagging, random forest, linear regression, etc. It will also aid in visualizing model performance through learning curves, accuracy, and loss across training rounds.

In [71]:
```
##First load in the relevant files. These should be saved on your local host and downl
# Install the necessary packages if you haven't already.
#We will be working with only cloud brightness and shortwave radiation.

import xarray as xr

reflected_SW_radiation = xr.open_dataset("C:/Users/kathe/Downloads/ReflectedSW_remappe
cloud_optical_depth=xr.open_dataset("C:/Users/kathe/Downloads/CloudOpticalDepth_remapp
cloud_top_temperature=xr.open_dataset("C:/Users/kathe/Downloads/CloudTopTemperature_re
downward_SW_radiation=xr.open_dataset("C:/Users/kathe/Downloads/DownwardSW_remapped.nc
total_precipitable_water=xr.open_dataset("C:/Users/kathe/Downloads/TotalPrecipWater_re
era5_meteorology=xr.open_dataset("C:/Users/kathe/Downloads/era5_data.nc")
era5_single_level=xr.open_dataset("C:/Users/kathe/Downloads/era5_single_levels.nc")
```

In [72]:
```
#Perform some basic functions to make sure the files have been loaded correctly. You s

#Print a summary of each dataset
print("Shortwave Radiation Dataset:")
print(reflected_SW_radiation)
print("\nCloud Brightness Dataset:")
print(cloud_optical_depth)
print("\nCloud Top Temperature Dataset:")
print(cloud_top_temperature)
print("\nDownward Shortwave Radiation Dataset:")
print(downward_SW_radiation)
print("\nTotal Precipitable Water Dataset:")
print(total_precipitable_water)
print("\nERA5 Single Level Dataset:")
print(era5_single_level)
```

```
Shortwave Radiation Dataset:
<xarray.Dataset>
Dimensions:      (time: 30, latitude: 41, longitude: 81)
Coordinates:
  * time         (time) int64 92 93 94 95 96 97 98 ... 116 117 118 119 120 121
  * latitude     (latitude) float64 20.0 20.25 20.5 20.75 ... 29.5 29.75 30.0
  * longitude    (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
Data variables:
    ReflectedSW  (time, latitude, longitude) float64 ...
Attributes:
    title:        Reflected Shortwave Remapped
    description:  Combined reflected shortwave data over time for Julian days...
    source:       GOES-16 Satellite Cloud and Moisture Imagery
    history:      Created on 2024-12-02.
    institution:  University of Washington
    references:   https://registry.opendata.aws/noaa-goes/
    time_units:   Julian days
    lat_units:    degrees_north
    lon_units:    degrees_east
    rsr_units:    W/m2

Cloud Brightness Dataset:
<xarray.Dataset>
Dimensions:            (time: 30, latitude: 41, longitude: 81)
Coordinates:
  * time               (time) int64 92 93 94 95 96 97 ... 117 118 119 120 121
  * latitude           (latitude) float64 20.0 20.25 20.5 ... 29.5 29.75 30.0
  * longitude          (longitude) float64 -65.0 -64.75 -64.5 ... -45.25 -45.0
Data variables:
    CloudOpticalDepth  (time, latitude, longitude) float64 ...
Attributes:
    title:        Cloud Optical Depth Remapped
    description:  Combined cloud optical depth data over time for Julian days...
    source:       GOES-16 Satellite Cloud and Moisture Imagery
    history:      Created on 2024-12-02.
    institution:  University of Washington
    references:   https://registry.opendata.aws/noaa-goes/
    time_units:   Julian days
    lat_units:    degrees_north
    lon_units:    degrees_east
    cod_units:    dimensionless

Cloud Top Temperature Dataset:
<xarray.Dataset>
Dimensions:             (time: 30, latitude: 41, longitude: 81)
Coordinates:
  * time                (time) int64 92 93 94 95 96 97 ... 117 118 119 120 121
  * latitude            (latitude) float64 20.0 20.25 20.5 ... 29.5 29.75 30.0
  * longitude           (longitude) float64 -65.0 -64.75 -64.5 ... -45.25 -45.0
Data variables:
    CloudTopTemperature  (time, latitude, longitude) float64 ...
Attributes:
    title:        Cloud Top Temperature Remapped
    description:  Combined cloud top temperature data over time for Julian da...
    source:       GOES-16 Satellite Cloud and Moisture Imagery
    history:      Created on 2024-12-02.
    institution:  University of Washington
    references:   https://registry.opendata.aws/noaa-goes/
    time_units:   Julian days
    lat_units:    degrees_north
```

```
    lon_units:    degrees_east
    temp_units:   K

Downward Shortwave Radiation Dataset:
<xarray.Dataset>
Dimensions:     (time: 30, latitude: 41, longitude: 81)
Coordinates:
  * time        (time) int64 92 93 94 95 96 97 98 ... 116 117 118 119 120 121
  * latitude    (latitude) float64 20.0 20.25 20.5 20.75 ... 29.5 29.75 30.0
  * longitude   (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
Data variables:
    DownwardSW  (time, latitude, longitude) float64 ...
Attributes:
    title:        Downward Shortwave Remapped
    description:  Combined downward shortwave data over time for Julian days ...
    source:       GOES-16 Satellite Cloud and Moisture Imagery
    history:      Created on 2024-12-02.
    institution:  University of Washington
    references:   https://registry.opendata.aws/noaa-goes/
    time_units:   Julian days
    lat_units:    degrees_north
    lon_units:    degrees_east
    dsr_units:    W/m2

Total Precipitable Water Dataset:
<xarray.Dataset>
Dimensions:           (time: 30, latitude: 41, longitude: 81)
Coordinates:
  * time              (time) int64 92 93 94 95 96 97 ... 116 117 118 119 120 121
  * latitude          (latitude) float64 20.0 20.25 20.5 ... 29.5 29.75 30.0
  * longitude         (longitude) float64 -65.0 -64.75 -64.5 ... -45.25 -45.0
Data variables:
    TotalPrecipWater  (time, latitude, longitude) float64 ...
Attributes:
    title:        Total Precip Water Remapped
    description:  Combined total precipitable water data over time for Julian...
    source:       GOES-16 Satellite Cloud and Moisture Imagery
    history:      Created on 2024-12-02.
    institution:  University of Washington
    references:   https://registry.opendata.aws/noaa-goes/
    time_units:   Julian days
    lat_units:    degrees_north
    lon_units:    degrees_east
    tpw_units:    mm

ERA5 Single Level Dataset:
<xarray.Dataset>
Dimensions:     (valid_time: 1650, latitude: 41, longitude: 81)
Coordinates:
    number      int64 ...
  * valid_time  (valid_time) datetime64[ns] 2014-04-01T13:00:00 ... 2024-04-3...
  * latitude    (latitude) float64 30.0 29.75 29.5 29.25 ... 20.5 20.25 20.0
  * longitude   (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
    expver      (valid_time) object ...
Data variables:
    u10         (valid_time, latitude, longitude) float32 ...
    v10         (valid_time, latitude, longitude) float32 ...
    t2m         (valid_time, latitude, longitude) float32 ...
    sst         (valid_time, latitude, longitude) float32 ...
    slhf        (valid_time, latitude, longitude) float32 ...
```

```
    sshf        (valid_time, latitude, longitude) float32 ...
    hcc         (valid_time, latitude, longitude) float32 ...
    lcc         (valid_time, latitude, longitude) float32 ...
    tcc         (valid_time, latitude, longitude) float32 ...
Attributes:
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2024-11-20T20:22 GRIB to CDM+CF via cfgrib-0.9.1...
```

In [73]:

```python
#Filter the era5 data to contain only 15:00 UTC and the year 2020

import pandas as pd


era5_single_level=xr.open_dataset("C:/Users/kathe/Downloads/era5_single_levels.nc")


# Convert `valid_time` to pandas DatetimeIndex if necessary
era5_single_level['valid_time'] = pd.to_datetime(era5_single_level.valid_time.values)

# Filter for 2020 and 15:00 UTC

filtered_single_level = era5_single_level.sel(
    valid_time=era5_single_level.valid_time[
        (era5_single_level.valid_time.dt.year == 2020) &
        (era5_single_level.valid_time.dt.hour == 15)
    ]
)

# Check the filtered data
print(filtered_single_level)
```

```
<xarray.Dataset>
Dimensions:     (valid_time: 30, latitude: 41, longitude: 81)
Coordinates:
    number      int64 ...
  * valid_time  (valid_time) datetime64[ns] 2020-04-01T15:00:00 ... 2020-04-3...
  * latitude    (latitude) float64 30.0 29.75 29.5 29.25 ... 20.5 20.25 20.0
  * longitude   (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
    expver      (valid_time) object ...
Data variables:
    u10         (valid_time, latitude, longitude) float32 ...
    v10         (valid_time, latitude, longitude) float32 ...
    t2m         (valid_time, latitude, longitude) float32 ...
    sst         (valid_time, latitude, longitude) float32 ...
    slhf        (valid_time, latitude, longitude) float32 ...
    sshf        (valid_time, latitude, longitude) float32 ...
    hcc         (valid_time, latitude, longitude) float32 ...
    lcc         (valid_time, latitude, longitude) float32 ...
    tcc         (valid_time, latitude, longitude) float32 ...
Attributes:
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2024-11-20T20:22 GRIB to CDM+CF via cfgrib-0.9.1...
```

In [ ]:
```
#Check for 15:00 UTC only and the year 2020
print(filtered_single_level.valid_time.values)
```

```
['2020-04-01T15:00:00.000000000' '2020-04-02T15:00:00.000000000'
 '2020-04-03T15:00:00.000000000' '2020-04-04T15:00:00.000000000'
 '2020-04-05T15:00:00.000000000' '2020-04-06T15:00:00.000000000'
 '2020-04-07T15:00:00.000000000' '2020-04-08T15:00:00.000000000'
 '2020-04-09T15:00:00.000000000' '2020-04-10T15:00:00.000000000'
 '2020-04-11T15:00:00.000000000' '2020-04-12T15:00:00.000000000'
 '2020-04-13T15:00:00.000000000' '2020-04-14T15:00:00.000000000'
 '2020-04-15T15:00:00.000000000' '2020-04-16T15:00:00.000000000'
 '2020-04-17T15:00:00.000000000' '2020-04-18T15:00:00.000000000'
 '2020-04-19T15:00:00.000000000' '2020-04-20T15:00:00.000000000'
 '2020-04-21T15:00:00.000000000' '2020-04-22T15:00:00.000000000'
 '2020-04-23T15:00:00.000000000' '2020-04-24T15:00:00.000000000'
 '2020-04-25T15:00:00.000000000' '2020-04-26T15:00:00.000000000'
 '2020-04-27T15:00:00.000000000' '2020-04-28T15:00:00.000000000'
 '2020-04-29T15:00:00.000000000' '2020-04-30T15:00:00.000000000']
['2020-04-01T15:00:00.000000000' '2020-04-02T15:00:00.000000000'
 '2020-04-03T15:00:00.000000000' '2020-04-04T15:00:00.000000000'
 '2020-04-05T15:00:00.000000000' '2020-04-06T15:00:00.000000000'
 '2020-04-07T15:00:00.000000000' '2020-04-08T15:00:00.000000000'
 '2020-04-09T15:00:00.000000000' '2020-04-10T15:00:00.000000000'
 '2020-04-11T15:00:00.000000000' '2020-04-12T15:00:00.000000000'
 '2020-04-13T15:00:00.000000000' '2020-04-14T15:00:00.000000000'
 '2020-04-15T15:00:00.000000000' '2020-04-16T15:00:00.000000000'
 '2020-04-17T15:00:00.000000000' '2020-04-18T15:00:00.000000000'
 '2020-04-19T15:00:00.000000000' '2020-04-20T15:00:00.000000000'
 '2020-04-21T15:00:00.000000000' '2020-04-22T15:00:00.000000000'
 '2020-04-23T15:00:00.000000000' '2020-04-24T15:00:00.000000000'
 '2020-04-25T15:00:00.000000000' '2020-04-26T15:00:00.000000000'
 '2020-04-27T15:00:00.000000000' '2020-04-28T15:00:00.000000000'
 '2020-04-29T15:00:00.000000000' '2020-04-30T15:00:00.000000000']
```

In [74]:
```
# #Save the filtered data to a new file


# filtered_single_level.to_netcdf('filtered_era5_single_level_2020_15UTC.nc')
```

In [75]:
```
#List the variable names in each dataset so we can understand all of the variables we

print("\nShortwave Radiation Dataset:")
print(list(reflected_SW_radiation.data_vars))
print("\nVariables in Cloud Optical Depth Dataset:")
print(list(cloud_optical_depth.data_vars))
print("\nVariables in Cloud Top Temperature Dataset:")
print(list(cloud_top_temperature.data_vars))
print("\nVariables in Downward Shortwave Radiation Dataset:")
print(list(downward_SW_radiation.data_vars))
print("\nVariables in Total Precipitable Water Dataset:")
print(list(total_precipitable_water.data_vars))
print("\nVariables in ERA5 Single Level Dataset:")
print(list(filtered_single_level.data_vars))
```

```
Shortwave Radiation Dataset:
['ReflectedSW']

Variables in Cloud Optical Depth Dataset:
['CloudOpticalDepth']

Variables in Cloud Top Temperature Dataset:
['CloudTopTemperature']

Variables in Downward Shortwave Radiation Dataset:
['DownwardSW']

Variables in Total Precipitable Water Dataset:
['TotalPrecipWater']

Variables in ERA5 Single Level Dataset:
['u10', 'v10', 't2m', 'sst', 'slhf', 'sshf', 'hcc', 'lcc', 'tcc']
```

Variables in ERA5 Meteorology Dataset: ['number', 'expver', 'cc', 'r', 'clwc', 't', 'u', 'v', 'w']

number=Represents the ensemble member identifier expver=Experiment version or identifier cc=cloud fraction r=relative humidity (%) clcw=cloud liquid water content t=temperature(K) u=wind component (m/s) E-W v=wind component (m/s) N-S w=vertical velocity

Variables in ERA5 Single Level Dataset: ['u10', 'v10', 't2m', 'sst', 'slhf', 'sshf', 'hcc', 'lcc', 'tcc']

u10=10-meter zonal wind component (m/s) v10=10-meter zonal wind component (m/s) t2m=2 meter temperature sst=sea surface temperature sshf=surface sensible heat flux W/m2 hcc=high cloud cover fraction lcc=low cloud cover fraction tcc=total cloud cover fraction

In [76]:
```
#Check to make sure the dimensions of each dataset are correct and match up.
print(reflected_SW_radiation.dims)
print(cloud_optical_depth.dims)
print(filtered_single_level.dims)
print(cloud_top_temperature.dims)
print(downward_SW_radiation.dims)
print(total_precipitable_water.dims)
```

```
Frozen({'time': 30, 'latitude': 41, 'longitude': 81})
Frozen({'time': 30, 'latitude': 41, 'longitude': 81})
Frozen({'valid_time': 30, 'latitude': 41, 'longitude': 81})
Frozen({'time': 30, 'latitude': 41, 'longitude': 81})
Frozen({'time': 30, 'latitude': 41, 'longitude': 81})
Frozen({'time': 30, 'latitude': 41, 'longitude': 81})
```

In [77]:
```
#The ERA5 and GOES datasets have different labeld time columns. We need these labels t
filtered_single_level = filtered_single_level.rename({'valid_time': 'time'})
```

In [78]:
```
print(filtered_single_level)
```

```
<xarray.Dataset>
Dimensions:    (time: 30, latitude: 41, longitude: 81)
Coordinates:
    number     int64 ...
  * time       (time) datetime64[ns] 2020-04-01T15:00:00 ... 2020-04-30T15:00:00
  * latitude   (latitude) float64 30.0 29.75 29.5 29.25 ... 20.5 20.25 20.0
  * longitude  (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
    expver     (time) object ...
Data variables:
    u10        (time, latitude, longitude) float32 ...
    v10        (time, latitude, longitude) float32 ...
    t2m        (time, latitude, longitude) float32 ...
    sst        (time, latitude, longitude) float32 ...
    slhf       (time, latitude, longitude) float32 ...
    sshf       (time, latitude, longitude) float32 ...
    hcc        (time, latitude, longitude) float32 ...
    lcc        (time, latitude, longitude) float32 ...
    tcc        (time, latitude, longitude) float32 ...
Attributes:
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2024-11-20T20:22 GRIB to CDM+CF via cfgrib-0.9.1...
```

In [79]:

```
#Double check the data type of each variable in each column of ERA5. They need to be c
for var in filtered_single_level.data_vars:
    print(f"{var}: {filtered_single_level[var].dtype}")
```

```
u10: float32
v10: float32
t2m: float32
sst: float32
slhf: float32
sshf: float32
hcc: float32
lcc: float32
tcc: float32
```

In [80]:

```
#Make sure everything is in float64 format. If not, convert it.
print(filtered_single_level.coords)
filtered_single_level = filtered_single_level.drop_vars(['expver', 'number'])
df_single_level = filtered_single_level.to_dataframe().reset_index()
print(df_single_level.info())
```

```
Coordinates:
    number     int64 ...
  * time       (time) datetime64[ns] 2020-04-01T15:00:00 ... 2020-04-30T15:00:00
  * latitude   (latitude) float64 30.0 29.75 29.5 29.25 ... 20.5 20.25 20.0
  * longitude  (longitude) float64 -65.0 -64.75 -64.5 ... -45.5 -45.25 -45.0
    expver     (time) object ...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99630 entries, 0 to 99629
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   time       99630 non-null  datetime64[ns]
 1   latitude   99630 non-null  float64
 2   longitude  99630 non-null  float64
 3   u10        99630 non-null  float32
 4   v10        99630 non-null  float32
 5   t2m        99630 non-null  float32
 6   sst        99630 non-null  float32
 7   slhf       99630 non-null  float32
 8   sshf       99630 non-null  float32
 9   hcc        99630 non-null  float32
 10  lcc        99630 non-null  float32
 11  tcc        99630 non-null  float32
dtypes: datetime64[ns](1), float32(9), float64(2)
memory usage: 5.7 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99630 entries, 0 to 99629
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   time       99630 non-null  datetime64[ns]
 1   latitude   99630 non-null  float64
 2   longitude  99630 non-null  float64
 3   u10        99630 non-null  float32
 4   v10        99630 non-null  float32
 5   t2m        99630 non-null  float32
 6   sst        99630 non-null  float32
 7   slhf       99630 non-null  float32
 8   sshf       99630 non-null  float32
 9   hcc        99630 non-null  float32
 10  lcc        99630 non-null  float32
 11  tcc        99630 non-null  float32
dtypes: datetime64[ns](1), float32(9), float64(2)
memory usage: 5.7 MB
None
```

In [81]:

```python
#From here on out, this dataset will be converted into a pandas dataframe. This will a
# basic data analysis on the dataset and make it easier for machine learning models to

#Load the pandas package if not already loaded

import pandas as pd

# Convert each xarray dataset to pandas DataFrame

df_shortwave = reflected_SW_radiation.to_dataframe().reset_index()
df_brightness = cloud_optical_depth.to_dataframe().reset_index()
df_cloud_top_temp = cloud_top_temperature.to_dataframe().reset_index()
```

```
df_downward_SW = downward_SW_radiation.to_dataframe().reset_index()
df_precipitable_water = total_precipitable_water.to_dataframe().reset_index()
```

In [82]:
```
df_shortwave['time'] = pd.to_datetime(df_shortwave['time'], format='%j') + pd.DateOffs
df_brightness['time'] = pd.to_datetime(df_brightness['time'], format='%j') + pd.DateOf
df_cloud_top_temp['time'] = pd.to_datetime(df_cloud_top_temp['time'], format='%j') + p
df_downward_SW['time'] = pd.to_datetime(df_downward_SW['time'], format='%j') + pd.Date
df_precipitable_water['time'] = pd.to_datetime(df_precipitable_water['time'], format='
```

```
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\pandas\core\arrays\datet
imes.py:760: PerformanceWarning: Non-vectorized DateOffset being applied to Series or
DatetimeIndex.
  warnings.warn(
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\pandas\core\arrays\datet
imes.py:760: PerformanceWarning: Non-vectorized DateOffset being applied to Series or
DatetimeIndex.
  warnings.warn(
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\pandas\core\arrays\datet
imes.py:760: PerformanceWarning: Non-vectorized DateOffset being applied to Series or
DatetimeIndex.
  warnings.warn(
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\pandas\core\arrays\datet
imes.py:760: PerformanceWarning: Non-vectorized DateOffset being applied to Series or
DatetimeIndex.
  warnings.warn(
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\pandas\core\arrays\datet
imes.py:760: PerformanceWarning: Non-vectorized DateOffset being applied to Series or
DatetimeIndex.
  warnings.warn(
```

In [83]:
```
df_shortwave['time'] = df_shortwave['time'] + pd.Timedelta(hours=15)
df_brightness['time'] = df_brightness['time'] + pd.Timedelta(hours=15)
df_cloud_top_temp['time'] = df_cloud_top_temp['time'] + pd.Timedelta(hours=15)
df_downward_SW['time'] = df_downward_SW['time'] + pd.Timedelta(hours=15)
df_precipitable_water['time'] = df_precipitable_water['time'] + pd.Timedelta(hours=15)
```

In [84]:
```
print(df_shortwave['time'].head())
print(df_brightness['time'].head())
print(df_cloud_top_temp['time'].head())
print(df_downward_SW['time'].head())
print(df_precipitable_water['time'].head())
print(df_single_level['time'].head())
```

```
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
0   2020-04-01 15:00:00
1   2020-04-01 15:00:00
2   2020-04-01 15:00:00
3   2020-04-01 15:00:00
4   2020-04-01 15:00:00
Name: time, dtype: datetime64[ns]
```

In [85]:

```
print("Unique times in df_shortwave:", df_shortwave['time'].unique())
print("Unique times in df_brightness:", df_brightness['time'].unique())
print("Unique times in df_single_level:", df_single_level['time'].unique())
```

```
Unique times in df_shortwave: ['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.0
00000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000' '2020-05-01T15:00:00.000000000']
Unique times in df_brightness: ['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.
000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000' '2020-05-01T15:00:00.000000000']
Unique times in df_single_level: ['2020-04-01T15:00:00.000000000' '2020-04-02T15:00:0
0.000000000'
 '2020-04-03T15:00:00.000000000' '2020-04-04T15:00:00.000000000'
 '2020-04-05T15:00:00.000000000' '2020-04-06T15:00:00.000000000'
 '2020-04-07T15:00:00.000000000' '2020-04-08T15:00:00.000000000'
 '2020-04-09T15:00:00.000000000' '2020-04-10T15:00:00.000000000'
 '2020-04-11T15:00:00.000000000' '2020-04-12T15:00:00.000000000'
 '2020-04-13T15:00:00.000000000' '2020-04-14T15:00:00.000000000'
 '2020-04-15T15:00:00.000000000' '2020-04-16T15:00:00.000000000'
 '2020-04-17T15:00:00.000000000' '2020-04-18T15:00:00.000000000'
 '2020-04-19T15:00:00.000000000' '2020-04-20T15:00:00.000000000'
 '2020-04-21T15:00:00.000000000' '2020-04-22T15:00:00.000000000'
 '2020-04-23T15:00:00.000000000' '2020-04-24T15:00:00.000000000'
 '2020-04-25T15:00:00.000000000' '2020-04-26T15:00:00.000000000'
 '2020-04-27T15:00:00.000000000' '2020-04-28T15:00:00.000000000'
 '2020-04-29T15:00:00.000000000' '2020-04-30T15:00:00.000000000']
```

In [86]:

```
for df, name in zip([df_cloud_top_temp, df_downward_SW, df_precipitable_water, df_sing
                    ['df_cloud_top_temp', 'df_downward_SW', 'df_precipitable_water', '
    print(f"Unique times in {name}: {df['time'].unique()}")
```

```
Unique times in df_cloud_top_temp: ['2020-04-02T15:00:00.000000000' '2020-04-03T15:0
0:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000' '2020-05-01T15:00:00.000000000']
Unique times in df_downward_SW: ['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:0
0.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000' '2020-05-01T15:00:00.000000000']
Unique times in df_precipitable_water: ['2020-04-02T15:00:00.000000000' '2020-04-03T1
5:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000' '2020-05-01T15:00:00.000000000']
Unique times in df_single_level: ['2020-04-01T15:00:00.000000000' '2020-04-02T15:00:0
0.000000000'
 '2020-04-03T15:00:00.000000000' '2020-04-04T15:00:00.000000000'
 '2020-04-05T15:00:00.000000000' '2020-04-06T15:00:00.000000000'
 '2020-04-07T15:00:00.000000000' '2020-04-08T15:00:00.000000000'
 '2020-04-09T15:00:00.000000000' '2020-04-10T15:00:00.000000000'
 '2020-04-11T15:00:00.000000000' '2020-04-12T15:00:00.000000000'
 '2020-04-13T15:00:00.000000000' '2020-04-14T15:00:00.000000000'
 '2020-04-15T15:00:00.000000000' '2020-04-16T15:00:00.000000000'
 '2020-04-17T15:00:00.000000000' '2020-04-18T15:00:00.000000000'
 '2020-04-19T15:00:00.000000000' '2020-04-20T15:00:00.000000000'
 '2020-04-21T15:00:00.000000000' '2020-04-22T15:00:00.000000000'
```

```
 '2020-04-23T15:00:00.000000000' '2020-04-24T15:00:00.000000000'
 '2020-04-25T15:00:00.000000000' '2020-04-26T15:00:00.000000000'
 '2020-04-27T15:00:00.000000000' '2020-04-28T15:00:00.000000000'
 '2020-04-29T15:00:00.000000000' '2020-04-30T15:00:00.000000000']
```

In [87]:

```
merged_df = pd.merge(df_shortwave, df_brightness, on=["time", "latitude", "longitude"]
merged_df = pd.merge(merged_df, df_cloud_top_temp, on=["time", "latitude", "longitude"
merged_df = pd.merge(merged_df, df_downward_SW, on=["time", "latitude", "longitude"],
merged_df = pd.merge(merged_df, df_precipitable_water, on=["time", "latitude", "longit
merged_df = pd.merge(merged_df, df_single_level, on=["time", "latitude", "longitude"],

print(merged_df.info())
print(merged_df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96309 entries, 0 to 96308
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   time                 96309 non-null  datetime64[ns]
 1   latitude             96309 non-null  float64
 2   longitude            96309 non-null  float64
 3   ReflectedSW          89349 non-null  float64
 4   CloudOpticalDepth    41064 non-null  float64
 5   CloudTopTemperature  37777 non-null  float64
 6   DownwardSW           89349 non-null  float64
 7   TotalPrecipWater     48767 non-null  float64
 8   u10                  96309 non-null  float32
 9   v10                  96309 non-null  float32
 10  t2m                  96309 non-null  float32
 11  sst                  96309 non-null  float32
 12  slhf                 96309 non-null  float32
 13  sshf                 96309 non-null  float32
 14  hcc                  96309 non-null  float32
 15  lcc                  96309 non-null  float32
 16  tcc                  96309 non-null  float32
dtypes: datetime64[ns](1), float32(9), float64(7)
memory usage: 9.9 MB
None
                 time  latitude  longitude  ReflectedSW  CloudOpticalDepth  \
0 2020-04-02 15:00:00      20.0     -65.00          NaN                NaN
1 2020-04-02 15:00:00      20.0     -64.75          NaN                NaN
2 2020-04-02 15:00:00      20.0     -64.50          NaN                NaN
3 2020-04-02 15:00:00      20.0     -64.25          NaN                NaN
4 2020-04-02 15:00:00      20.0     -64.00          NaN                NaN

   CloudTopTemperature  DownwardSW  TotalPrecipWater       u10       v10  \
0                  NaN         NaN               NaN  1.470306  6.316940
1                  NaN         NaN               NaN  1.066010  6.272995
2                  NaN         NaN               NaN  0.687103  6.278854
3                  NaN         NaN               NaN  0.458588  6.209518
4                  NaN         NaN               NaN  0.342377  6.184128

          t2m         sst      slhf     sshf  hcc       lcc       tcc
0  299.085571  299.293945 -549491.0   -558.0  0.0  0.052216  0.052216
1  299.003540  299.239258 -558323.0  -1582.0  0.0  0.028900  0.028900
2  298.855103  299.279297 -569907.0  -7470.0  0.0  0.000000  0.000000
3  298.741821  299.268555 -564275.0 -10862.0  0.0  0.000000  0.000000
4  298.655884  299.296875 -569331.0 -14702.0  0.0  0.000000  0.000000
```

In [ ]:
```
# df_shortwave['time'] = pd.to_datetime(df_shortwave['time'])
# df_brightness['time'] = pd.to_datetime(df_brightness['time'])
# df_cloud_top_temp['time'] = pd.to_datetime(df_cloud_top_temp['time'])
# df_downward_SW['time'] = pd.to_datetime(df_downward_SW['time'])
# df_precipitable_water['time'] = pd.to_datetime(df_precipitable_water['time'])
# df_single_level['time'] = pd.to_datetime(df_single_level['time'])
```

In [ ]:
```
# merged_df = pd.merge(df_shortwave, df_brightness, on=["time", "latitude", "longitude
# merged_df = pd.merge(merged_df, df_cloud_top_temp, on=["time", "latitude", "longitud
# merged_df = pd.merge(merged_df, df_downward_SW, on=["time", "latitude", "longitude"]
# merged_df = pd.merge(merged_df, df_precipitable_water, on=["time", "latitude", "long
# merged_df = pd.merge(merged_df, df_single_level, on=["time", "latitude", "longitude"
```

In [46]:
```
# print(merged_df.info())
# print(merged_df.head())
```

In [47]:
```
# daily_mean_df = merged_df.groupby('time').mean().reset_index()
# print(daily_mean_df.head())
```

In [88]:
```
# Define the April time range
start_date = "2020-04-01T15:00:00"
end_date = "2020-04-30T15:00:00"

# Filter each DataFrame for April dates
df_shortwave = df_shortwave[(df_shortwave['time'] >= start_date) & (df_shortwave['time
df_brightness = df_brightness[(df_brightness['time'] >= start_date) & (df_brightness['
df_cloud_top_temp = df_cloud_top_temp[(df_cloud_top_temp['time'] >= start_date) & (df_
df_downward_SW = df_downward_SW[(df_downward_SW['time'] >= start_date) & (df_downward_
df_precipitable_water = df_precipitable_water[(df_precipitable_water['time'] >= start_
df_single_level = df_single_level[(df_single_level['time'] >= start_date) & (df_single
```

In [89]:
```
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp,
                     df_downward_SW, df_precipitable_water, df_single_level],
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp',
                     'df_downward_SW', 'df_precipitable_water', 'df_single_level']):
    print(f"{name} - Unique Times:")
    print(df['time'].unique())
```

```
df_shortwave - Unique Times:
['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000']
df_brightness - Unique Times:
['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000']
df_cloud_top_temp - Unique Times:
['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000']
df_downward_SW - Unique Times:
['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
```

```
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000']
df_precipitable_water - Unique Times:
['2020-04-02T15:00:00.000000000' '2020-04-03T15:00:00.000000000'
 '2020-04-04T15:00:00.000000000' '2020-04-05T15:00:00.000000000'
 '2020-04-06T15:00:00.000000000' '2020-04-07T15:00:00.000000000'
 '2020-04-08T15:00:00.000000000' '2020-04-09T15:00:00.000000000'
 '2020-04-10T15:00:00.000000000' '2020-04-11T15:00:00.000000000'
 '2020-04-12T15:00:00.000000000' '2020-04-13T15:00:00.000000000'
 '2020-04-14T15:00:00.000000000' '2020-04-15T15:00:00.000000000'
 '2020-04-16T15:00:00.000000000' '2020-04-17T15:00:00.000000000'
 '2020-04-18T15:00:00.000000000' '2020-04-19T15:00:00.000000000'
 '2020-04-20T15:00:00.000000000' '2020-04-21T15:00:00.000000000'
 '2020-04-22T15:00:00.000000000' '2020-04-23T15:00:00.000000000'
 '2020-04-24T15:00:00.000000000' '2020-04-25T15:00:00.000000000'
 '2020-04-26T15:00:00.000000000' '2020-04-27T15:00:00.000000000'
 '2020-04-28T15:00:00.000000000' '2020-04-29T15:00:00.000000000'
 '2020-04-30T15:00:00.000000000']
df_single_level - Unique Times:
['2020-04-01T15:00:00.000000000' '2020-04-02T15:00:00.000000000'
 '2020-04-03T15:00:00.000000000' '2020-04-04T15:00:00.000000000'
 '2020-04-05T15:00:00.000000000' '2020-04-06T15:00:00.000000000'
 '2020-04-07T15:00:00.000000000' '2020-04-08T15:00:00.000000000'
 '2020-04-09T15:00:00.000000000' '2020-04-10T15:00:00.000000000'
 '2020-04-11T15:00:00.000000000' '2020-04-12T15:00:00.000000000'
 '2020-04-13T15:00:00.000000000' '2020-04-14T15:00:00.000000000'
 '2020-04-15T15:00:00.000000000' '2020-04-16T15:00:00.000000000'
 '2020-04-17T15:00:00.000000000' '2020-04-18T15:00:00.000000000'
 '2020-04-19T15:00:00.000000000' '2020-04-20T15:00:00.000000000'
 '2020-04-21T15:00:00.000000000' '2020-04-22T15:00:00.000000000'
 '2020-04-23T15:00:00.000000000' '2020-04-24T15:00:00.000000000'
 '2020-04-25T15:00:00.000000000' '2020-04-26T15:00:00.000000000'
 '2020-04-27T15:00:00.000000000' '2020-04-28T15:00:00.000000000'
 '2020-04-29T15:00:00.000000000' '2020-04-30T15:00:00.000000000']
```

In [90]:

```python
merged_df = pd.merge(df_shortwave, df_brightness, on=["time", "latitude", "longitude"]
merged_df = pd.merge(merged_df, df_cloud_top_temp, on=["time", "latitude", "longitude"
merged_df = pd.merge(merged_df, df_downward_SW, on=["time", "latitude", "longitude"],
merged_df = pd.merge(merged_df, df_precipitable_water, on=["time", "latitude", "longit
merged_df = pd.merge(merged_df, df_single_level, on=["time", "latitude", "longitude"],

print(merged_df.info())
print(merged_df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96309 entries, 0 to 96308
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   time                 96309 non-null  datetime64[ns]
 1   latitude             96309 non-null  float64
 2   longitude            96309 non-null  float64
 3   ReflectedSW          89349 non-null  float64
 4   CloudOpticalDepth    41064 non-null  float64
 5   CloudTopTemperature  37777 non-null  float64
 6   DownwardSW           89349 non-null  float64
 7   TotalPrecipWater     48767 non-null  float64
 8   u10                  96309 non-null  float32
 9   v10                  96309 non-null  float32
 10  t2m                  96309 non-null  float32
 11  sst                  96309 non-null  float32
 12  slhf                 96309 non-null  float32
 13  sshf                 96309 non-null  float32
 14  hcc                  96309 non-null  float32
 15  lcc                  96309 non-null  float32
 16  tcc                  96309 non-null  float32
dtypes: datetime64[ns](1), float32(9), float64(7)
memory usage: 9.9 MB
None
                 time  latitude  longitude  ReflectedSW  CloudOpticalDepth  \
0 2020-04-02 15:00:00      20.0     -65.00          NaN                NaN
1 2020-04-02 15:00:00      20.0     -64.75          NaN                NaN
2 2020-04-02 15:00:00      20.0     -64.50          NaN                NaN
3 2020-04-02 15:00:00      20.0     -64.25          NaN                NaN
4 2020-04-02 15:00:00      20.0     -64.00          NaN                NaN

   CloudTopTemperature  DownwardSW  TotalPrecipWater       u10       v10  \
0                  NaN         NaN               NaN  1.470306  6.316940
1                  NaN         NaN               NaN  1.066010  6.272995
2                  NaN         NaN               NaN  0.687103  6.278854
3                  NaN         NaN               NaN  0.458588  6.209518
4                  NaN         NaN               NaN  0.342377  6.184128

         t2m         sst      slhf     sshf  hcc       lcc       tcc
0  299.085571  299.293945 -549491.0   -558.0  0.0  0.052216  0.052216
1  299.003540  299.239258 -558323.0  -1582.0  0.0  0.028900  0.028900
2  298.855103  299.279297 -569907.0  -7470.0  0.0  0.000000  0.000000
3  298.741821  299.268555 -564275.0 -10862.0  0.0  0.000000  0.000000
4  298.655884  299.296875 -569331.0 -14702.0  0.0  0.000000  0.000000
```

In [91]:

```
#Check lat and long are consistent across all the data
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp,
                     df_downward_SW, df_precipitable_water, df_single_level],
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp',
                     'df_downward_SW', 'df_precipitable_water', 'df_single_level']):
    print(f"{name} Latitude Range: {df['latitude'].min()} to {df['latitude'].max()}")
    print(f"{name} Longitude Range: {df['longitude'].min()} to {df['longitude'].max()}
```

```
df_shortwave Latitude Range: 20.0 to 30.0
df_shortwave Longitude Range: -65.0 to -45.0
df_brightness Latitude Range: 20.0 to 30.0
df_brightness Longitude Range: -65.0 to -45.0
df_cloud_top_temp Latitude Range: 20.0 to 30.0
df_cloud_top_temp Longitude Range: -65.0 to -45.0
df_downward_SW Latitude Range: 20.0 to 30.0
df_downward_SW Longitude Range: -65.0 to -45.0
df_precipitable_water Latitude Range: 20.0 to 30.0
df_precipitable_water Longitude Range: -65.0 to -45.0
df_single_level Latitude Range: 20.0 to 30.0
df_single_level Longitude Range: -65.0 to -45.0
```

In [92]:

```
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp,
                     df_downward_SW, df_precipitable_water, df_single_level],
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp',
                     'df_downward_SW', 'df_precipitable_water', 'df_single_level']):
    print(f"{name} Missing Values:")
    print(df[['time', 'latitude', 'longitude']].isnull().sum())
```

```
df_shortwave Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
df_brightness Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
df_cloud_top_temp Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
df_downward_SW Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
df_precipitable_water Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
df_single_level Missing Values:
time         0
latitude     0
longitude    0
dtype: int64
```

In [93]:

```
debug_df = pd.merge(df_shortwave, df_brightness, on=["time", "latitude", "longitude"],
print(debug_df['_merge'].value_counts())
print(debug_df[debug_df['_merge'] == 'left_only'].head())  # Rows only in df_shortwave
print(debug_df[debug_df['_merge'] == 'right_only'].head())  # Rows only in df_brightne
```

```
both           96309
left_only          0
right_only         0
Name: _merge, dtype: int64
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, CloudOpticalDepth, _merge]
Index: []
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, CloudOpticalDepth, _merge]
Index: []
```

In [94]:

```
debug_df = pd.merge(df_shortwave, df_cloud_top_temp, on=["time", "latitude", "longituc
print(debug_df['_merge'].value_counts())
print(debug_df[debug_df['_merge'] == 'left_only'].head())  # Rows only in df_shortwave
print(debug_df[debug_df['_merge'] == 'right_only'].head())  # Rows only in df_cloud_tc
```

```
both           96309
left_only          0
right_only         0
Name: _merge, dtype: int64
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, CloudTopTemperature, _merge]
Index: []
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, CloudTopTemperature, _merge]
Index: []
```

In [95]:

```
debug_df = pd.merge(df_shortwave, df_downward_SW, on=["time", "latitude", "longitude"]
print(debug_df['_merge'].value_counts())
print(debug_df[debug_df['_merge'] == 'left_only'].head())  # Rows only in df_shortwave
print(debug_df[debug_df['_merge'] == 'right_only'].head())  # Rows only in df_downwarc
```

```
both           96309
left_only          0
right_only         0
Name: _merge, dtype: int64
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, DownwardSW, _merge]
Index: []
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, DownwardSW, _merge]
Index: []
```

In [96]:

```
debug_df = pd.merge(df_shortwave, df_precipitable_water, on=["time", "latitude", "long
print(debug_df['_merge'].value_counts())
print(debug_df[debug_df['_merge'] == 'left_only'].head())  # Rows only in df_shortwave
print(debug_df[debug_df['_merge'] == 'right_only'].head())  # Rows only in df_precipit
```

```
both           96309
left_only          0
right_only         0
Name: _merge, dtype: int64
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, TotalPrecipWater, _merge]
Index: []
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, TotalPrecipWater, _merge]
Index: []
```

In [97]:

```
debug_df = pd.merge(df_shortwave, df_single_level, on=["time", "latitude", "longitude"
print(debug_df['_merge'].value_counts())
```

```
print(debug_df[debug_df['_merge'] == 'left_only'].head())  # Rows only in df_shortwave
print(debug_df[debug_df['_merge'] == 'right_only'].head())  # Rows only in df_single_l
```

```
both          96309
right_only     3321
left_only         0
Name: _merge, dtype: int64
Empty DataFrame
Columns: [time, latitude, longitude, ReflectedSW, u10, v10, t2m, sst, slhf, sshf, hc
c, lcc, tcc, _merge]
Index: []
                     time  latitude  longitude  ReflectedSW       u10  \
96309 2020-04-01 15:00:00      30.0     -65.00          NaN  7.607880
96310 2020-04-01 15:00:00      30.0     -64.75          NaN  8.104950
96311 2020-04-01 15:00:00      30.0     -64.50          NaN  8.227020
96312 2020-04-01 15:00:00      30.0     -64.25          NaN  8.074677
96313 2020-04-01 15:00:00      30.0     -64.00          NaN  7.761200

            v10         t2m         sst      slhf    sshf       hcc       lcc  \
96309  9.157684  295.778931  295.776367 -699337.0  5658.0  0.904999  0.252502
96310  8.940887  295.798462  295.924805 -718089.0 -2214.0  0.831940  0.305511
96311  8.727020  295.755493  295.929688 -690633.0 -5094.0  0.732025  0.335480
96312  8.584442  295.704712  295.880859 -656777.0 -5670.0  0.672913  0.325531
96313  8.401825  295.630493  295.745117 -614665.0 -2342.0  0.614960  0.349487

            tcc      _merge
96309  0.912506  right_only
96310  0.885468  right_only
96311  0.811523  right_only
96312  0.769440  right_only
96313  0.755463  right_only
```

In [98]:

```
print(df_shortwave['time'].head())
print(df_shortwave['time'].dtype)
print(df_brightness['time'].head())
print(df_brightness['time'].dtype)
print(df_cloud_top_temp['time'].head())
print(df_cloud_top_temp['time'].dtype)
print(df_downward_SW['time'].head())
print(df_downward_SW['time'].dtype)
print(df_precipitable_water['time'].head())
print(df_precipitable_water['time'].dtype)
print(df_single_level['time'].head())
print(df_single_level['time'].dtype)
```

```
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
0   2020-04-02 15:00:00
1   2020-04-02 15:00:00
2   2020-04-02 15:00:00
3   2020-04-02 15:00:00
4   2020-04-02 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
0   2020-04-01 15:00:00
1   2020-04-01 15:00:00
2   2020-04-01 15:00:00
3   2020-04-01 15:00:00
4   2020-04-01 15:00:00
Name: time, dtype: datetime64[ns]
datetime64[ns]
```

In [99]:

```
check_date = pd.Timestamp('2020-04-02 15:00:00')

for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp,
                     df_downward_SW, df_precipitable_water, df_single_level],
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp',
                     'df_downward_SW', 'df_precipitable_water', 'df_single_level']):
    print(f"{name} - Contains April 2nd:")
    print(check_date in df['time'].unique())
```

```
df_shortwave - Contains April 2nd:
True
df_brightness - Contains April 2nd:
True
df_cloud_top_temp - Contains April 2nd:
True
df_downward_SW - Contains April 2nd:
True
df_precipitable_water - Contains April 2nd:
True
df_single_level - Contains April 2nd:
True
```

In [100…

```
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp,
                     df_downward_SW, df_precipitable_water, df_single_level],
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp',
                     'df_downward_SW', 'df_precipitable_water', 'df_single_level']):
    print(f"{name} - Date Range: {df['time'].min()} to {df['time'].max()}")
```

```
df_shortwave - Date Range: 2020-04-02 15:00:00 to 2020-04-30 15:00:00
df_brightness - Date Range: 2020-04-02 15:00:00 to 2020-04-30 15:00:00
df_cloud_top_temp - Date Range: 2020-04-02 15:00:00 to 2020-04-30 15:00:00
df_downward_SW - Date Range: 2020-04-02 15:00:00 to 2020-04-30 15:00:00
df_precipitable_water - Date Range: 2020-04-02 15:00:00 to 2020-04-30 15:00:00
df_single_level - Date Range: 2020-04-01 15:00:00 to 2020-04-30 15:00:00
```

In [101…

```
df_single_level = df_single_level[
    (df_single_level['time'] >= '2020-04-02 15:00:00') &
    (df_single_level['time'] <= '2020-04-30 15:00:00')
]
```

In [102…

```
print(f"df_single_level - Date Range After Filtering: {df_single_level['time'].min()}
```

```
df_single_level - Date Range After Filtering: 2020-04-02 15:00:00 to 2020-04-30 15:0
0:00
```

In [103…

```
matching_keys = pd.merge(
    df_shortwave[['time', 'latitude', 'longitude']],
    df_single_level[['time', 'latitude', 'longitude']],
    on=["time", "latitude", "longitude"],
    how="inner"
)
print(f"Number of matching keys: {len(matching_keys)}")
```

```
Number of matching keys: 96309
```

In [104…

```
for df, name in zip([df_brightness, df_cloud_top_temp, df_downward_SW, df_precipitable
                    ['df_brightness', 'df_cloud_top_temp', 'df_downward_SW', 'df_preci
    matching_keys = pd.merge(
        df_shortwave[['time', 'latitude', 'longitude']],
        df[['time', 'latitude', 'longitude']],
        on=["time", "latitude", "longitude"],
        how="inner"
    )
    print(f"{name} - Number of matching keys: {len(matching_keys)}")
```

```
df_brightness - Number of matching keys: 96309
df_cloud_top_temp - Number of matching keys: 96309
df_downward_SW - Number of matching keys: 96309
df_precipitable_water - Number of matching keys: 96309
```

In [105…

```
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp, df_downward_SW, d
                     ['df_shortwave', 'df_brightness', 'df_cloud_top_temp', 'df_downwar
    print(f"{name} Missing Values:")
    print(df.isnull().sum())
    print("-" * 50)
```

```
df_shortwave Missing Values:
time              0
latitude          0
longitude         0
ReflectedSW    6960
dtype: int64
--------------------------------------------------
df_brightness Missing Values:
time                     0
latitude                 0
longitude                0
CloudOpticalDepth    55245
dtype: int64
--------------------------------------------------
df_cloud_top_temp Missing Values:
time                       0
latitude                   0
longitude                  0
CloudTopTemperature    58532
dtype: int64
--------------------------------------------------
df_downward_SW Missing Values:
time             0
latitude         0
longitude        0
DownwardSW    6960
dtype: int64
--------------------------------------------------
df_precipitable_water Missing Values:
time                    0
latitude                0
longitude               0
TotalPrecipWater    47542
dtype: int64
--------------------------------------------------
```

There is missing data. We are going to fill the missing data by taking the mean of the 3 rows surrounding empty rows of data and fill it.

In [106…

```
# Forward-fill example:
df_shortwave['ReflectedSW'].fillna(method='ffill', inplace=True)

# Mean imputation example:
df_brightness['CloudOpticalDepth'].fillna(df_brightness['CloudOpticalDepth'].mean(), i

# Apply to all datasets with missing values:
datasets = [df_shortwave, df_brightness, df_cloud_top_temp, df_downward_SW, df_precipi
for df in datasets:
    for col in df.columns[3:]:  # Impute only data columns (not time, latitude, longit
        df[col].fillna(df[col].mean(), inplace=True)  # Replace 'mean()' with appropri
```

In [107…

```
for df, name in zip([df_shortwave, df_brightness, df_cloud_top_temp, df_downward_SW, d
                    ['df_shortwave', 'df_brightness', 'df_cloud_top_temp', 'df_downwar
    print(f"{name} Missing Values After Imputation:")
    print(df.isnull().sum())
    print("-" * 50)
```

```
df_shortwave Missing Values After Imputation:
time           0
latitude       0
longitude      0
ReflectedSW    0
dtype: int64
--------------------------------------------------
df_brightness Missing Values After Imputation:
time                 0
latitude             0
longitude            0
CloudOpticalDepth    0
dtype: int64
--------------------------------------------------
df_cloud_top_temp Missing Values After Imputation:
time                   0
latitude               0
longitude              0
CloudTopTemperature    0
dtype: int64
--------------------------------------------------
df_downward_SW Missing Values After Imputation:
time          0
latitude      0
longitude     0
DownwardSW    0
dtype: int64
--------------------------------------------------
df_precipitable_water Missing Values After Imputation:
time                0
latitude            0
longitude           0
TotalPrecipWater    0
dtype: int64
--------------------------------------------------
```

In [108…

```
# Perform the merge step-by-step
merged_df = pd.merge(df_shortwave, df_brightness, on=["time", "latitude", "longitude"]
merged_df = pd.merge(merged_df, df_cloud_top_temp, on=["time", "latitude", "longitude"
merged_df = pd.merge(merged_df, df_downward_SW, on=["time", "latitude", "longitude"])
merged_df = pd.merge(merged_df, df_precipitable_water, on=["time", "latitude", "longit
merged_df = pd.merge(merged_df, df_single_level, on=["time", "latitude", "longitude"])
```

In [109…

```
print(merged_df.info())  # Summary of merged DataFrame
print(merged_df.head())  # Preview the first few rows
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96309 entries, 0 to 96308
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype         
---  ------               --------------  -----         
 0   time                 96309 non-null  datetime64[ns]
 1   latitude             96309 non-null  float64       
 2   longitude            96309 non-null  float64       
 3   ReflectedSW          96309 non-null  float64       
 4   CloudOpticalDepth    96309 non-null  float64       
 5   CloudTopTemperature  96309 non-null  float64       
 6   DownwardSW           96309 non-null  float64       
 7   TotalPrecipWater     96309 non-null  float64       
 8   u10                  96309 non-null  float32       
 9   v10                  96309 non-null  float32       
 10  t2m                  96309 non-null  float32       
 11  sst                  96309 non-null  float32       
 12  slhf                 96309 non-null  float32       
 13  sshf                 96309 non-null  float32       
 14  hcc                  96309 non-null  float32       
 15  lcc                  96309 non-null  float32       
 16  tcc                  96309 non-null  float32       
dtypes: datetime64[ns](1), float32(9), float64(7)
memory usage: 9.9 MB
None
                 time  latitude  longitude  ReflectedSW  CloudOpticalDepth  \
0 2020-04-02 15:00:00      20.0     -65.00   193.426418           8.252434   
1 2020-04-02 15:00:00      20.0     -64.75   193.426418           8.252434   
2 2020-04-02 15:00:00      20.0     -64.50   193.426418           8.252434   
3 2020-04-02 15:00:00      20.0     -64.25   193.426418           8.252434   
4 2020-04-02 15:00:00      20.0     -64.00   193.426418           8.252434   

   CloudTopTemperature  DownwardSW  TotalPrecipWater       u10       v10  \
0           258.802026  877.561519          26.81986  1.470306  6.316940   
1           258.802026  877.561519          26.81986  1.066010  6.272995   
2           258.802026  877.561519          26.81986  0.687103  6.278854   
3           258.802026  877.561519          26.81986  0.458588  6.209518   
4           258.802026  877.561519          26.81986  0.342377  6.184128   

          t2m         sst      slhf     sshf  hcc       lcc       tcc  
0  299.085571  299.293945 -549491.0   -558.0  0.0  0.052216  0.052216  
1  299.003540  299.239258 -558323.0  -1582.0  0.0  0.028900  0.028900  
2  298.855103  299.279297 -569907.0  -7470.0  0.0  0.000000  0.000000  
3  298.741821  299.268555 -564275.0 -10862.0  0.0  0.000000  0.000000  
4  298.655884  299.296875 -569331.0 -14702.0  0.0  0.000000  0.000000  
```

In [110…

```
for col in merged_df.columns:
    unique_values = merged_df[col].nunique()
    print(f"{col}: {unique_values} unique values")
```

```
time: 29 unique values
latitude: 41 unique values
longitude: 81 unique values
ReflectedSW: 42722 unique values
CloudOpticalDepth: 41029 unique values
CloudTopTemperature: 37778 unique values
DownwardSW: 45185 unique values
TotalPrecipWater: 48768 unique values
u10: 77551 unique values
v10: 75892 unique values
t2m: 27355 unique values
sst: 21997 unique values
slhf: 75422 unique values
sshf: 33495 unique values
hcc: 20360 unique values
lcc: 23391 unique values
tcc: 28952 unique values
```

In [111…

```
features = merged_df[['ReflectedSW', 'CloudOpticalDepth', 'CloudTopTemperature',
                      'DownwardSW', 'TotalPrecipWater', 'tcc']]
```

Now that the dataset is ready to go. We need to begin training the model on the first 20 days of April, before we can test the model on the last 10 days of April.

In [50]:

```
#This code randomly splits the data for the whole month for training. Do not use this
# from sklearn.model_selection import train_test_split

# # Define features for clustering
# features = merged_df[['ReflectedSW', 'CloudOpticalDepth', 'CloudTopTemperature',
#                        'DownwardSW', 'TotalPrecipWater', 'tcc']]

# # First Split: 80% for training/validation, 20% for testing
# train_val_data, test_data = train_test_split(features, test_size=0.2, random_state=4

# # Second Split: 90% of training/validation for training, 10% for validation
# train_data, val_data = train_test_split(train_val_data, test_size=0.1, random_state=

# # Print sizes to confirm
# print("Training data size:", train_data.shape)
# print("Validation data size:", val_data.shape)
# print("Test data size:", test_data.shape)
```

First we need to cluster the data because we expect unique clusters based on the data along with meaningful subgroups or patterns. First try K-means clustering.

In [112…

```
from sklearn.cluster import KMeans

# Select features for clustering (excluding time, latitude, longitude)
clustering_features = merged_df.drop(columns=['time', 'latitude', 'longitude'])

# Perform clustering (choose an appropriate number of clusters, e.g., 3)
kmeans = KMeans(n_clusters=3, random_state=42)
merged_df['Cluster'] = kmeans.fit_predict(clustering_features)

# Print cluster assignments
print(merged_df['Cluster'].value_counts())
```

```
1    41595
2    36934
0    17780
Name: Cluster, dtype: int64
```

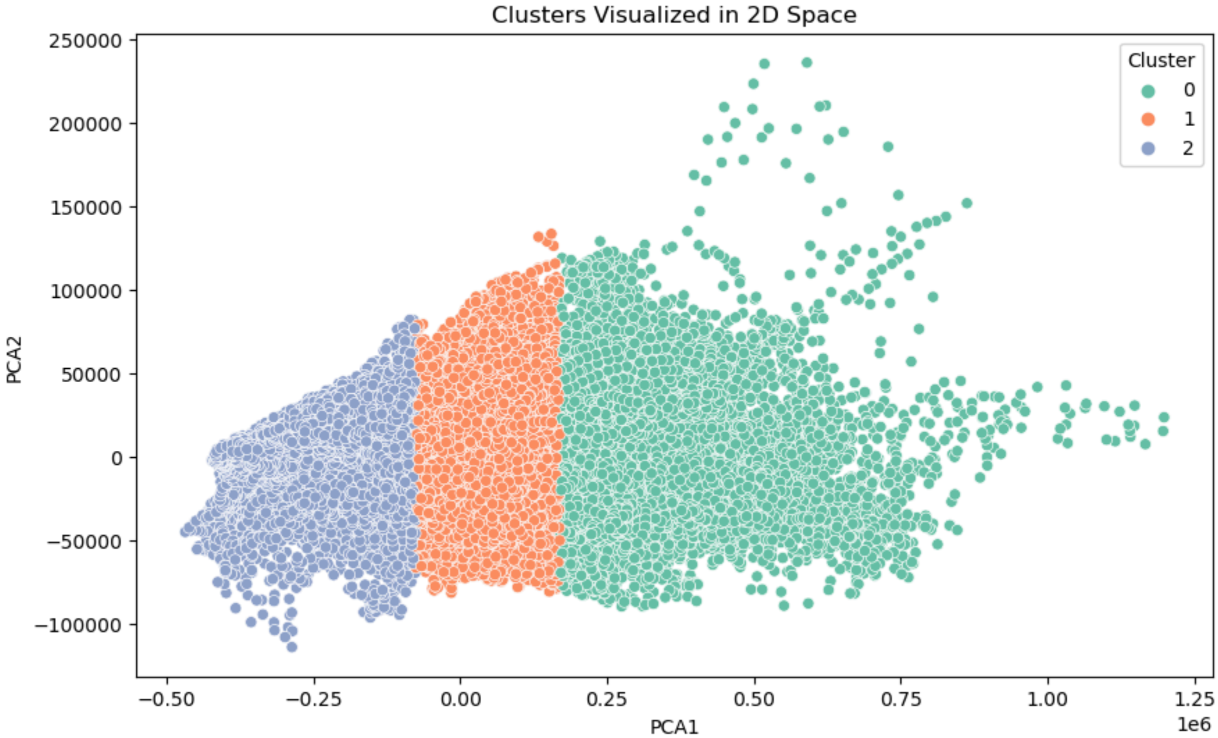
In [113…

```
# Access cluster centroids
centroids = kmeans.cluster_centers_
print("Cluster Centroids:")
print(pd.DataFrame(centroids, columns=clustering_features.columns))
```

```
Cluster Centroids:
   ReflectedSW  CloudOpticalDepth  CloudTopTemperature  DownwardSW  \
0   226.330986          10.211811           260.658027  839.068733
1   180.884591           7.993740           260.097111  895.770993
2   191.687477           7.598021           256.444100  875.603538

   TotalPrecipWater       u10       v10         t2m         sst  \
0         25.972770 -2.502995 -0.468947  296.250183  297.475627
1         26.582611 -1.714938  1.527403  296.616660  297.569460
2         27.496726  0.219832  2.865938  296.468364  297.005864

            slhf          sshf       hcc       lcc       tcc
0 -741719.975466 -47923.293342  0.092092  0.308808  0.414218
1 -476341.512254 -28972.125042  0.175534  0.244773  0.414829
2 -252668.024404 -12656.063885  0.317420  0.226935  0.513547
```

In [114…

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Reduce dimensionality for visualization
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(clustering_features)
merged_df['PCA1'] = reduced_data[:, 0]
merged_df['PCA2'] = reduced_data[:, 1]

# Plot clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=merged_df, x='PCA1', y='PCA2', hue='Cluster', palette='Set2')
plt.title("Clusters Visualized in 2D Space")
plt.show()
```



In [115…

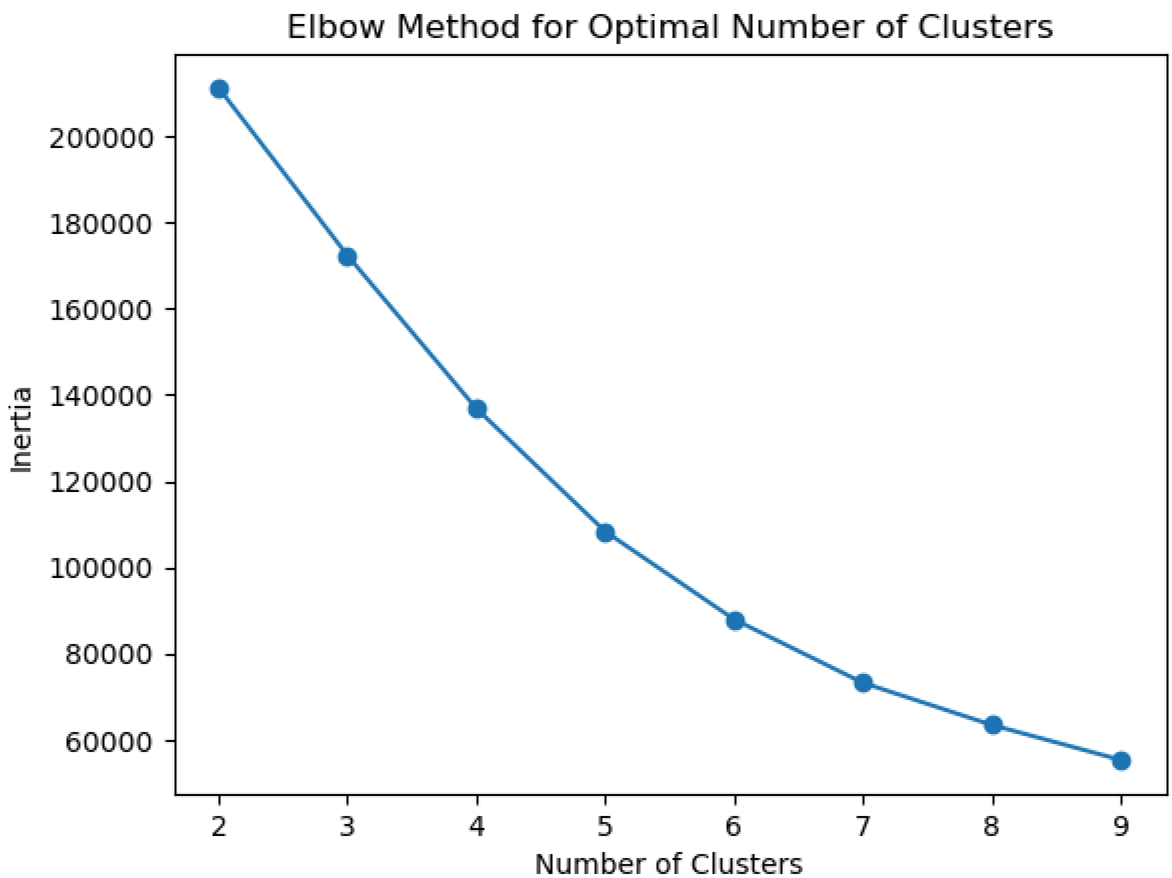
```
for cluster_id in merged_df['Cluster'].unique():
    cluster_data = merged_df[merged_df['Cluster'] == cluster_id]
    print(f"Cluster {cluster_id} Feature Means:")
    print(cluster_data[clustering_features.columns].mean())
    print("-" * 50)
```

```
Cluster 1 Feature Means:
ReflectedSW              180.902593
CloudOpticalDepth          7.993510
CloudTopTemperature      260.096240
DownwardSW               895.764509
TotalPrecipWater          26.582687
u10                       -1.719329
v10                        1.524528
t2m                      296.610901
sst                      297.573059
slhf                 -476596.843750
sshf                  -28987.371094
hcc                        0.175485
lcc                        0.244745
tcc                        0.414759
dtype: float64
--------------------------------------------------
Cluster 0 Feature Means:
ReflectedSW              226.416933
CloudOpticalDepth         10.217668
CloudTopTemperature      260.657945
DownwardSW               838.959035
TotalPrecipWater          25.973402
u10                       -2.502541
v10                       -0.470651
t2m                      296.248108
sst                      297.475922
slhf                 -741963.687500
sshf                  -47944.722656
hcc                        0.091997
lcc                        0.308870
tcc                        0.414218
dtype: float64
--------------------------------------------------
Cluster 2 Feature Means:
ReflectedSW              191.649115
CloudOpticalDepth          7.597972
CloudTopTemperature      256.451046
DownwardSW               875.644555
TotalPrecipWater          27.494447
u10                        0.220806
v10                        2.866152
t2m                      296.466339
sst                      297.008087
slhf                 -252846.703125
sshf                  -12670.930664
hcc                        0.317236
lcc                        0.227018
tcc                        0.513480
dtype: float64
--------------------------------------------------
```

In [116…

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Try different numbers of clusters
inertia = []
for k in range(2, 10):  # Range of clusters to test
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
```

```
    inertia.append(kmeans.inertia_)

# Plot inertia to find the 'elbow'
plt.plot(range(2, 10), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.show()
```



In [117…

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)  # X_train should be your features before cl



from sklearn.metrics import silhouette_score

for k in range(2, 10):  # Test different numbers of clusters
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    print(f"Silhouette Score for {k} clusters: {score}")
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11528\1924045630.py in <module>
     10     kmeans = KMeans(n_clusters=k, random_state=42)
     11     labels = kmeans.fit_predict(X_scaled)
---> 12     score = silhouette_score(X_scaled, labels)
     13     print(f"Silhouette Score for {k} clusters: {score}")
     14 

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\cluster
\_unsupervised.py in silhouette_score(X, labels, metric, sample_size, random_state, *
*kwds)
    115         else:
    116             X, labels = X[indices], labels[indices]
--> 117     return np.mean(silhouette_samples(X, labels, metric=metric, **kwds))
    118 
    119 

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\cluster
\_unsupervised.py in silhouette_samples(X, labels, metric, **kwds)
    231         _silhouette_reduce, labels=labels, label_freqs=label_freqs
    232     )
--> 233     results = zip(*pairwise_distances_chunked(X, reduce_func=reduce_func, **k
wds))
    234     intra_clust_dists, inter_clust_dists = results
    235     intra_clust_dists = np.concatenate(intra_clust_dists)

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\pairwis
e.py in pairwise_distances_chunked(X, Y, reduce_func, metric, n_jobs, working_memory,
**kwds)
   1715         else:
   1716             X_chunk = X[sl]
-> 1717         D_chunk = pairwise_distances(X_chunk, Y, metric=metric, n_jobs=n_job
s, **kwds)
   1718         if (X is Y or Y is None) and PAIRWISE_DISTANCE_FUNCTIONS.get(
   1719             metric, None

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\pairwis
e.py in pairwise_distances(X, Y, metric, n_jobs, force_all_finite, **kwds)
   1887         func = partial(distance.cdist, metric=metric, **kwds)
   1888 
-> 1889     return _parallel_pairwise(X, Y, func, n_jobs, **kwds)
   1890 
   1891 

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\pairwis
e.py in _parallel_pairwise(X, Y, func, n_jobs, **kwds)
   1428 
   1429     if effective_n_jobs(n_jobs) == 1:
-> 1430         return func(X, Y, **kwds)
   1431 
   1432     # enforce a threading backend to prevent data communication overhead

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\pairwis
e.py in euclidean_distances(X, Y, Y_norm_squared, squared, X_norm_squared)
    328             )
    329 
--> 330     return _euclidean_distances(X, Y, X_norm_squared, Y_norm_squared, square
d)
    331 
```

```
    332

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\metrics\pairwis
e.py in _euclidean_distances(X, Y, X_norm_squared, Y_norm_squared, squared)
    369     else:
    370         # if dtype is already float64, no need to chunk and upcast
--> 371         distances = -2 * safe_sparse_dot(X, Y.T, dense_output=True)
    372         distances += XX
    373         distances += YY

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\utils\extmath.py
in safe_sparse_dot(a, b, dense_output)
    151             ret = np.dot(a, b)
    152     else:
--> 153         ret = a @ b
    154 
    155     if (

KeyboardInterrupt:
```

Try nonspherical clustering.

In [118…

```python
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=10)  # Adjust eps and min_samples as needed
dbscan_labels = dbscan.fit_predict(X_scaled)

# Check number of clusters (ignoring noise, labeled as -1)
print(f"Number of clusters (excluding noise): {len(set(dbscan_labels)) - (1 if -1 in d
```

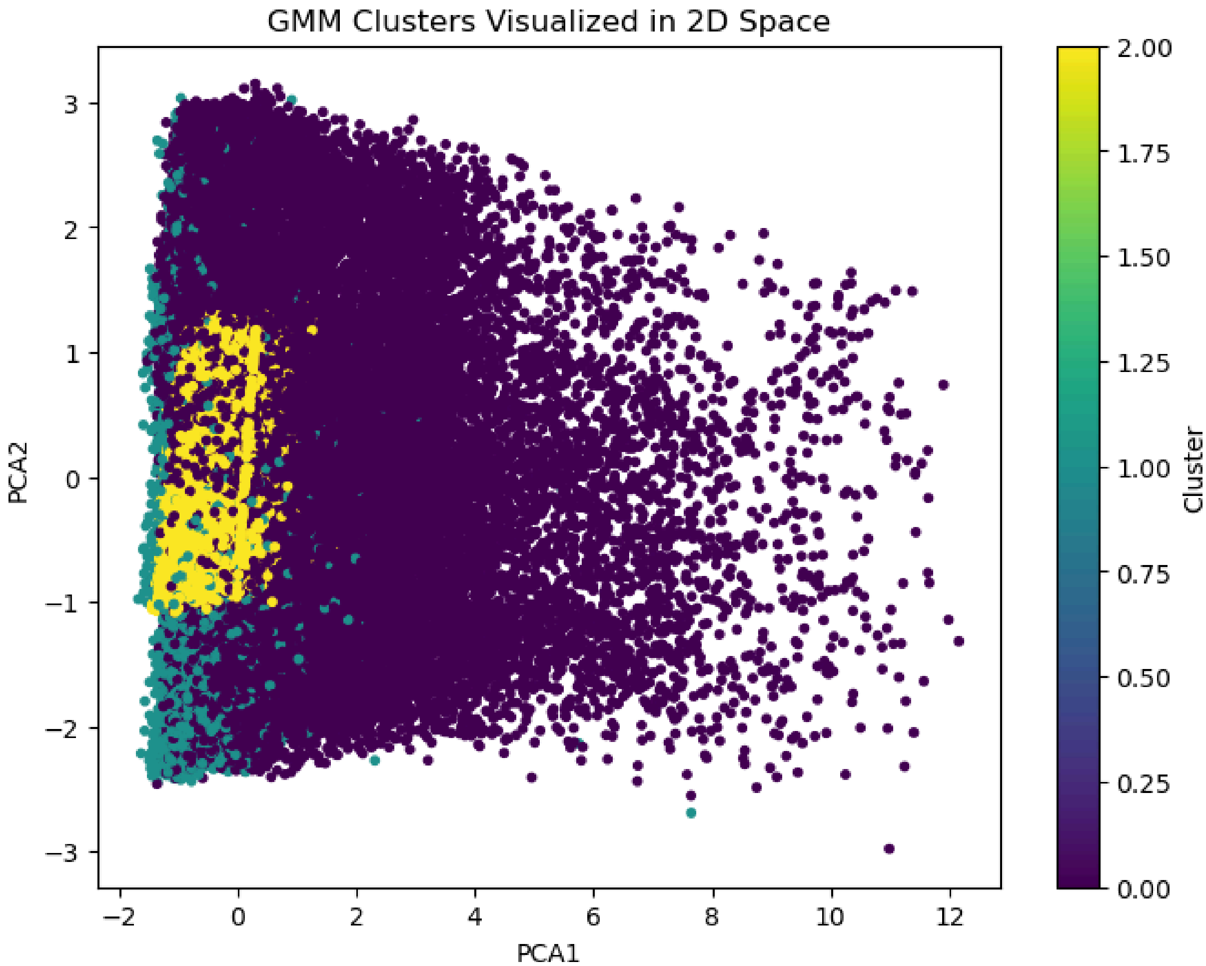
```
Number of clusters (excluding noise): 14
```

Try hierarchal clustering

In [119…

```python
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, random_state=42)  # Adjust n_components
gmm_labels = gmm.fit_predict(X_scaled)
merged_df['GMM_Cluster'] = gmm_labels
print(merged_df['GMM_Cluster'].value_counts())
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Perform PCA to reduce to 2 dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=gmm_labels, cmap='viridis', s=10)
plt.title("GMM Clusters Visualized in 2D Space")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.colorbar(label="Cluster")
plt.show()
```

```
2    52992
0    36593
1     6724
Name: GMM_Cluster, dtype: int64
```



If the clustering is poor, meaning there is overlap in the clusters and they are not distinct groups, try clustering after training the model instead.

Now split the data to begin training.

In [120…

```
# Sort the data by time
merged_df = merged_df.sort_values(by='time')

# Split the data into training and testing based on time
train_data = merged_df[merged_df['time'] < '2020-04-21']  # First 19 days (April 2 to
test_data = merged_df[merged_df['time'] >= '2020-04-21']  # Last 10 days (April 21 to

# Further split the training data into training and validation sets
from sklearn.model_selection import train_test_split
train_val_data, val_data = train_test_split(train_data, test_size=0.1, random_state=42

# Print details of the splits
print("Training data time range:", train_data['time'].min(), "to", train_data['time'].
print("Training data size:", train_data.shape)
print("Validation data time range:", val_data['time'].min(), "to", val_data['time'].ma
print("Validation data size:", val_data.shape)
print("Test data time range:", test_data['time'].min(), "to", test_data['time'].max())
print("Test data size:", test_data.shape)
```

```
Training data time range: 2020-04-02 15:00:00 to 2020-04-20 15:00:00
Training data size: (63099, 21)
Validation data time range: 2020-04-02 15:00:00 to 2020-04-20 15:00:00
Validation data size: (6310, 21)
Test data time range: 2020-04-21 15:00:00 to 2020-04-30 15:00:00
Test data size: (33210, 21)
```

We need to scale the data now using our cluster feature.

In [121…
```
feature_columns = ['ReflectedSW', 'CloudOpticalDepth', 'CloudTopTemperature',
                   'DownwardSW', 'TotalPrecipWater', 'tcc']

# Split features for training, validation, and testing
X_train = train_data[feature_columns]
X_val = val_data[feature_columns]
X_test = test_data[feature_columns]
```

In [122…
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit the scaler on the training data and transform all sets
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

In [123…
```
print("First few rows of scaled training data:")
print(X_train_scaled[:5])
```

```
First few rows of scaled training data:
[[-0.08084254 -0.06177183 -0.08926311  0.11361274  0.00635961 -1.24823985]
 [-0.66822003 -0.06177183 -0.08926311  0.44836796 -1.26986559 -0.18788789]
 [-0.63941808 -0.06177183 -0.08926311  0.4482365   0.00635961 -0.2213116 ]
 [-0.69117162 -0.06177183 -0.08926311  0.44810504 -1.12199029 -0.06810295]
 [-0.69072158 -0.06177183 -0.08926311  0.46808829 -1.06606102 -0.23582189]]
```

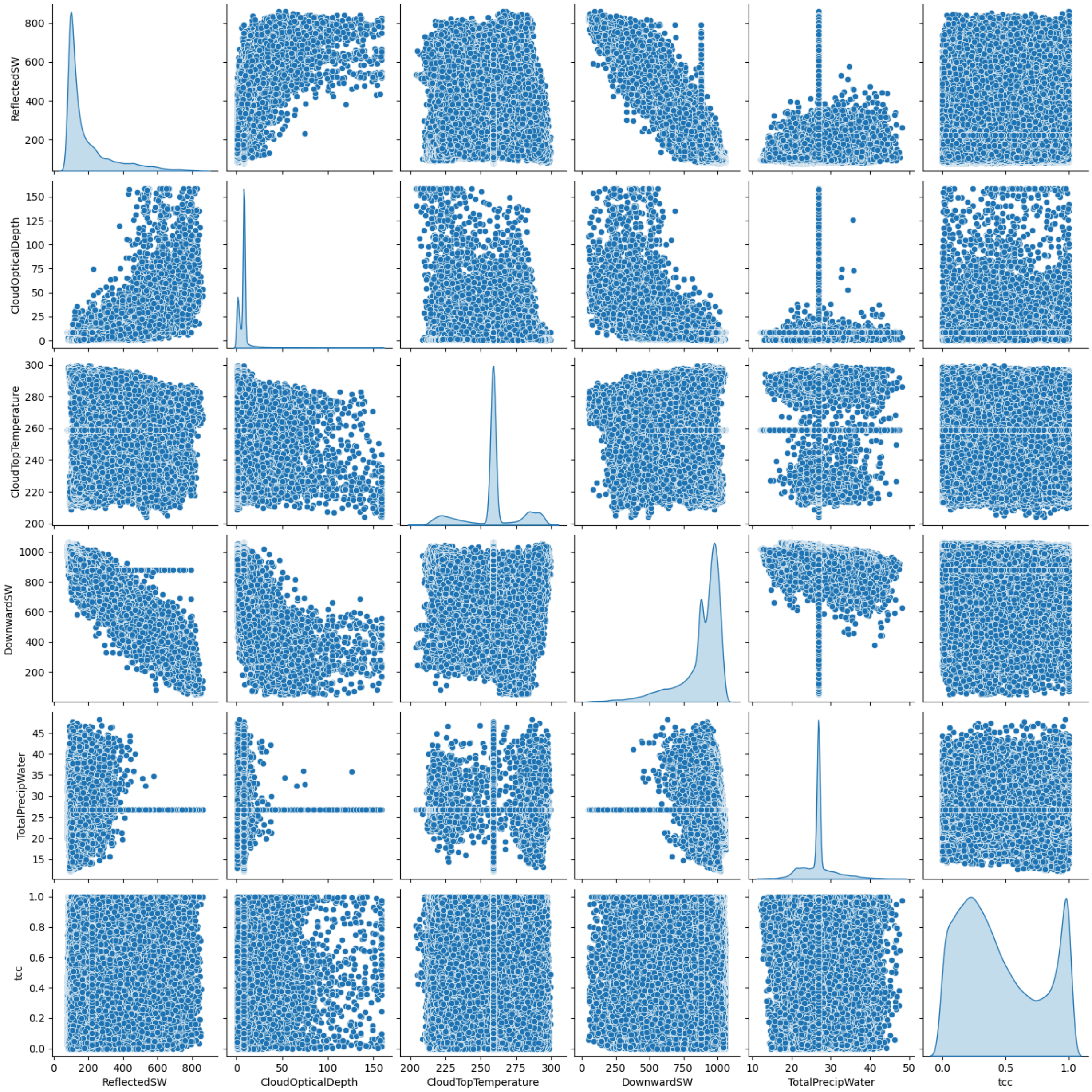
In [ ]:
```
# from sklearn.preprocessing import StandardScaler

# scaler = StandardScaler()
# train_data_scaled = scaler.fit_transform(train_data)
# val_data_scaled = scaler.transform(val_data)
# test_data_scaled = scaler.transform(test_data)
```

In [94]:
```
# # Check for missing values
# print(features.isnull().sum())

# # Check for infinity values
# print((features == float('inf')).sum())
# print((features == float('-inf')).sum())
```

Perform some quick exploratory data analysis, like scatter plots and histrograms just to visualize the data.

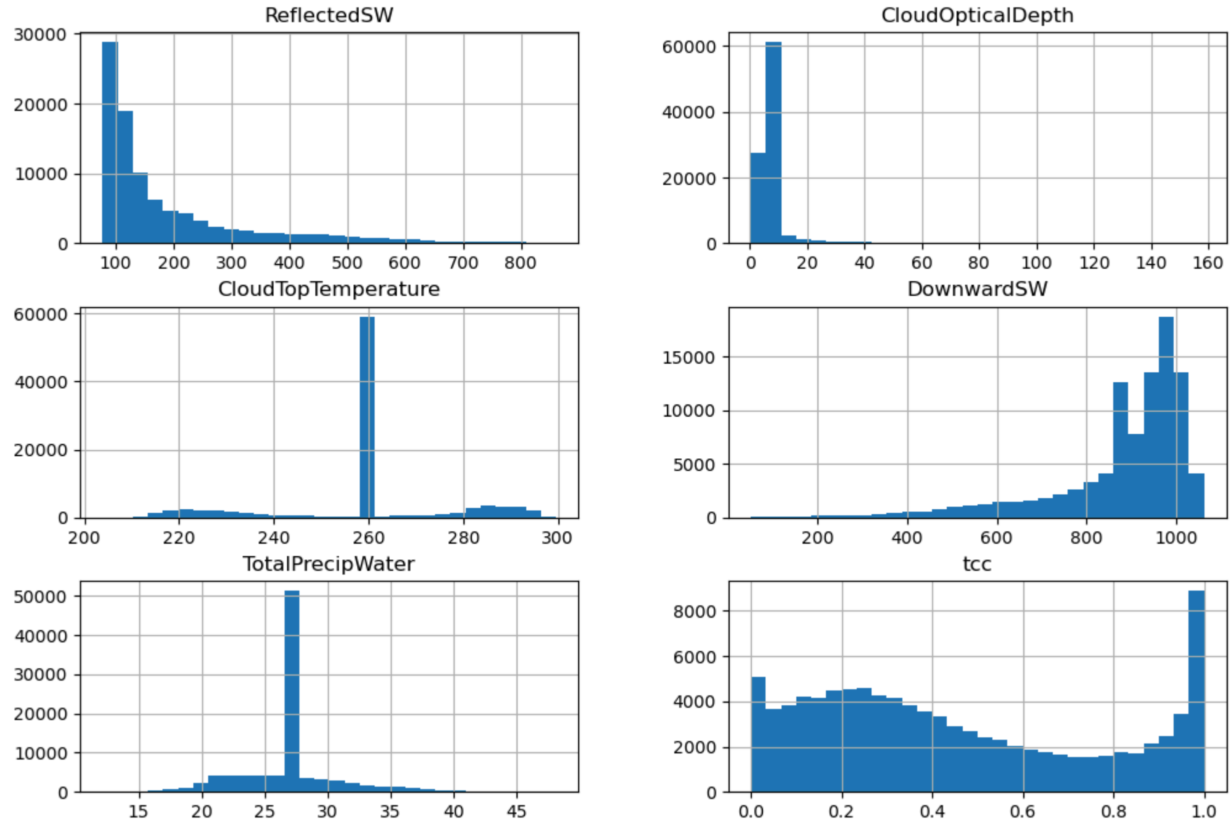
In [124…
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.pairplot(features, diag_kind='kde')
plt.show()
```



In [125…

```
features.hist(figsize=(12, 8), bins=30)
plt.show()
```



Begin RandomForest

In [126…

```
feature_columns = ['ReflectedSW', 'CloudOpticalDepth', 'CloudTopTemperature',
                   'DownwardSW', 'TotalPrecipWater']  # Features for the model

X_train = train_data[feature_columns]
y_train = train_data['tcc']  # Target variable
X_val = val_data[feature_columns]
y_val = val_data['tcc']
X_test = test_data[feature_columns]
y_test = test_data['tcc']
```

In [127…

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

In [128…

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state=42, n_estimators=100)
rf.fit(X_train_scaled, y_train)
```

Out[128]:

```
RandomForestRegressor(random_state=42)
```

In [129…

```
from sklearn.metrics import mean_squared_error, r2_score

# Validation
y_val_pred = rf.predict(X_val_scaled)
val_mse = mean_squared_error(y_val, y_val_pred)
```

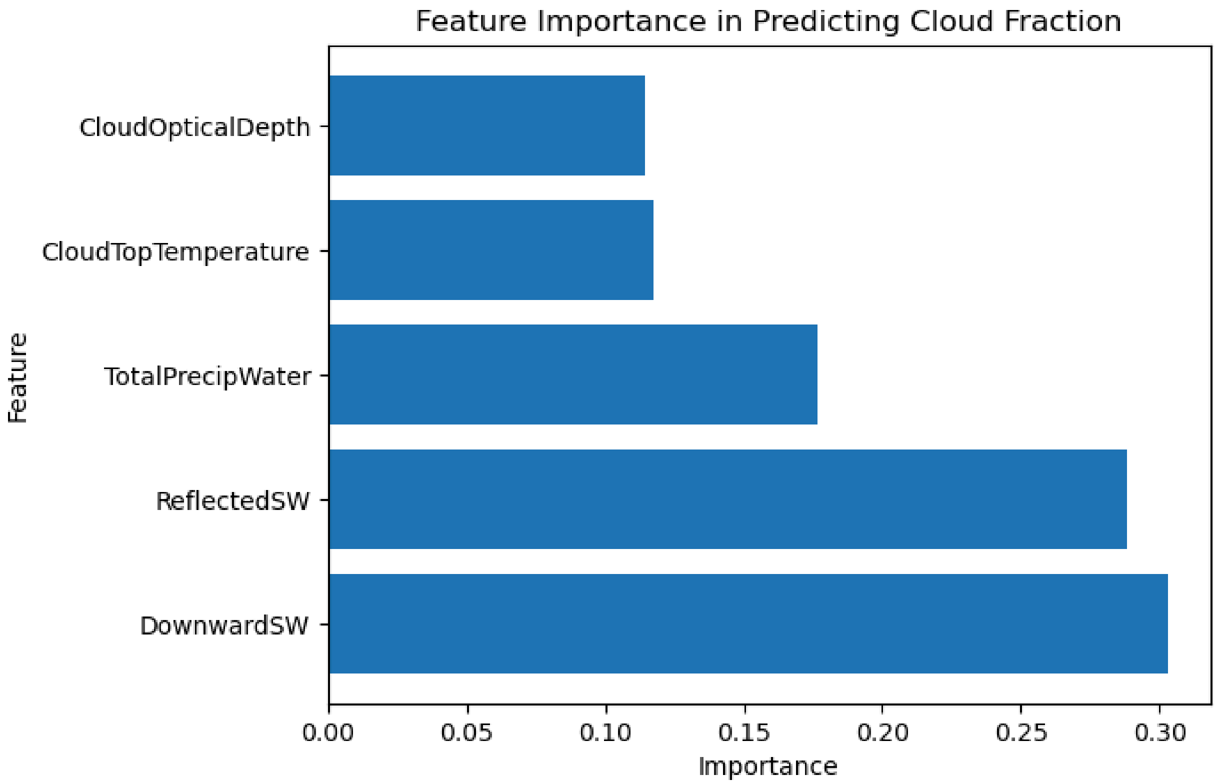
```
val_r2 = r2_score(y_val, y_val_pred)
print(f"Validation MSE: {val_mse}")
print(f"Validation R²: {val_r2}")

# Testing
y_test_pred = rf.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f"Test MSE: {test_mse}")
print(f"Test R²: {test_r2}")
```

```
Validation MSE: 0.016697000112504034
Validation R²: 0.818885230122995
Test MSE: 0.12347903833970987
Test R²: -0.13520913414546198
```

Show feature importance

In [130…

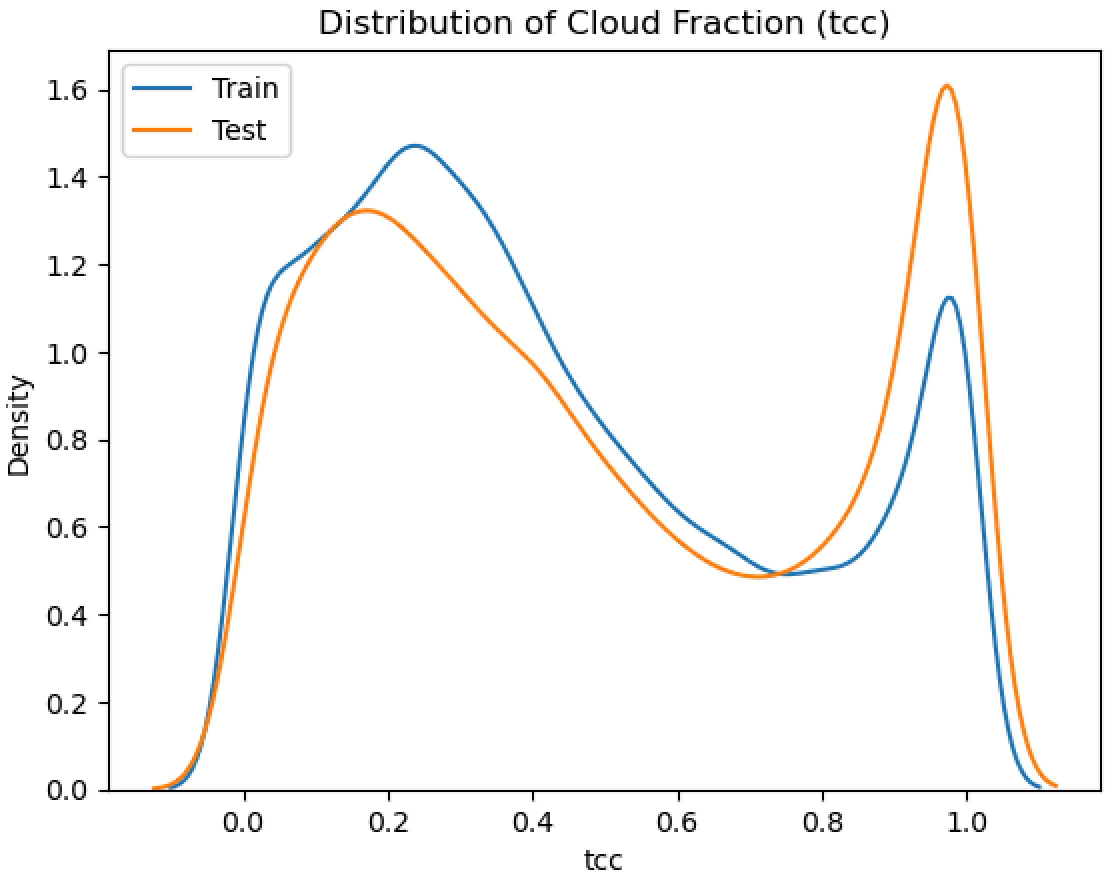
```
import matplotlib.pyplot as plt
import pandas as pd

# Get feature importances
feature_importances = rf.feature_importances_

# Create a DataFrame for visualization
importance_df = pd.DataFrame({
    'Feature': feature_columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importance in Predicting Cloud Fraction")
plt.show()
```



Examine distributional shifts in the data over the month long time period.

In [131…

```
# Compare the distributions of tcc between train and test
sns.kdeplot(train_data['tcc'], label='Train')
sns.kdeplot(test_data['tcc'], label='Test')
plt.title("Distribution of Cloud Fraction (tcc)")
plt.legend()
plt.show()
```



Tune the RandomForest model if the peaks for cloud fraction do not match for test and train.

In [132…

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='r2',
    verbose=2,
    n_jobs=-1
)

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Output best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best CV R² Score:", grid_search.best_score_)
```

```
# Train final model with best parameters
best_rf = grid_search.best_estimator_
y_test_pred = best_rf.predict(X_test_scaled)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

```
exception calling callback for <Future at 0x22c9fd4a8b0 state=finished raised BrokenP
rocessPool>
joblib.externals.loky.process_executor._RemoteTraceback:
"""
Traceback (most recent call last):
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals
\loky\process_executor.py", line 407, in _process_worker
    call_item = call_queue.get(block=True, timeout=timeout)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\multiprocessing\queues.py", li
ne 122, in get
    return _ForkingPickler.loads(res)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\__init_
_.py", line 82, in <module>
    from .base import clone
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\base.p
y", line 17, in <module>
    from .utils import _IS_32BIT
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\utils\__
init__.py", line 21, in <module>
    from scipy.sparse import issparse
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\scipy\sparse\__i
nit__.py", line 267, in <module>
    from ._csr import *
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\scipy\sparse\_cs
r.py", line 10, in <module>
    from ._sparsetools import (csr_tocsc, csr_tobsr, csr_count_blocks,
ImportError: numpy.core.multiarray failed to import
"""

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals
\loky\_base.py", line 625, in _invoke_callbacks
    callback(self)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.
py", line 359, in __call__
    self.parallel.dispatch_next()
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.
py", line 794, in dispatch_next
    if not self.dispatch_one_batch(self._original_iterator):
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.
py", line 861, in dispatch_one_batch
    self._dispatch(tasks)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.
py", line 779, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\_parallel
_backends.py", line 531, in apply_async
    future = self._workers.submit(SafeFunction(func))
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals
\loky\reusable_executor.py", line 177, in submit
    return super(_ReusablePoolExecutor, self).submit(
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals
\loky\process_executor.py", line 1115, in submit
    raise self._flags.broken
joblib.externals.loky.process_executor.BrokenProcessPool: A task has failed to un-ser
ialize. Please ensure that the arguments of the function are all picklable.
```

```
---------------------------------------------------------------------------
_RemoteTraceback                          Traceback (most recent call last)
_RemoteTraceback:
"""
Traceback (most recent call last):
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals
\loky\process_executor.py", line 407, in _process_worker
    call_item = call_queue.get(block=True, timeout=timeout)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\multiprocessing\queues.py", li
ne 122, in get
    return _ForkingPickler.loads(res)
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\__init_
_.py", line 82, in <module>
    from .base import clone
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\base.p
y", line 17, in <module>
    from .utils import _IS_32BIT
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\utils\__
init__.py", line 21, in <module>
    from scipy.sparse import issparse
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\scipy\sparse\__i
nit__.py", line 267, in <module>
    from ._csr import *
  File "c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\scipy\sparse\_cs
r.py", line 10, in <module>
    from ._sparsetools import (csr_tocsc, csr_tobsr, csr_count_blocks,
ImportError: numpy.core.multiarray failed to import
"""

The above exception was the direct cause of the following exception:

BrokenProcessPool                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11528\785054326.py in <module>
     21 
     22 # Fit the model
---> 23 grid_search.fit(X_train_scaled, y_train)
     24 
     25 # Output best parameters and score

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\model_selection
\_search.py in fit(self, X, y, groups, **fit_params)
    889                 return results
    890 
--> 891             self._run_search(evaluate_candidates)
    892 
    893             # multimetric is determined here because in the case of a callabl
e

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\model_selection
\_search.py in _run_search(self, evaluate_candidates)
   1390     def _run_search(self, evaluate_candidates):
   1391         """Search all candidates in param_grid"""
-> 1392         evaluate_candidates(ParameterGrid(self.param_grid))
   1393 
   1394 

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\sklearn\model_selection
\_search.py in evaluate_candidates(candidate_params, cv, more_results)
    836                     )
    837 
```

```
--> 838                 out = parallel(
    839                     delayed(_fit_and_score)(
    840                         clone(base_estimator),

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in __
call__(self, iterable)
   1054
   1055             with self._backend.retrieval_context():
-> 1056                 self.retrieve()
   1057             # Make sure that we get a last message telling us we are done
   1058             elapsed_time = time.time() - self._start_time

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in re
trieve(self)
    933             try:
    934                 if getattr(self._backend, 'supports_timeout', False):
--> 935                     self._output.extend(job.get(timeout=self.timeout))
    936                 else:
    937                     self._output.extend(job.get())

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\_parallel_backend
s.py in wrap_future_result(future, timeout)
    540         AsyncResults.get from multiprocessing."""
    541         try:
--> 542             return future.result(timeout=timeout)
    543         except CfTimeoutError as e:
    544             raise TimeoutError from e

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\concurrent\futures\_base.py in result
(self, timeout)
    444                     raise CancelledError()
    445                 elif self._state == FINISHED:
--> 446                     return self.__get_result()
    447                 else:
    448                     raise TimeoutError()

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\concurrent\futures\_base.py in __get_r
esult(self)
    389         if self._exception:
    390             try:
--> 391                 raise self._exception
    392             finally:
    393                 # Break a reference cycle with the exception in self._excepti
on

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals\loky\_b
ase.py in _invoke_callbacks(self)
    623         for callback in self._done_callbacks:
    624             try:
--> 625                 callback(self)
    626             except BaseException:
    627                 LOGGER.exception('exception calling callback for %r', self)

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in __
call__(self, out)
    357         with self.parallel._lock:
    358             if self.parallel._original_iterator is not None:
--> 359                 self.parallel.dispatch_next()
    360
    361
```

```
c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in di
spatch_next(self)
    792
    793         """
--> 794         if not self.dispatch_one_batch(self._original_iterator):
    795             self._iterating = False
    796             self._original_iterator = None

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in di
spatch_one_batch(self, iterator)
    859                 return False
    860             else:
--> 861                 self._dispatch(tasks)
    862                 return True
    863

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\parallel.py in _d
ispatch(self, batch)
    777         with self._lock:
    778             job_idx = len(self._jobs)
--> 779             job = self._backend.apply_async(batch, callback=cb)
    780             # A job can complete so quickly than its callback is
    781             # called before we get here, causing self._jobs to

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\_parallel_backend
s.py in apply_async(self, func, callback)
    529     def apply_async(self, func, callback=None):
    530         """Schedule a func to be run"""
--> 531         future = self._workers.submit(SafeFunction(func))
    532         future.get = functools.partial(self.wrap_future_result, future)
    533         if callback is not None:

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals\loky\re
usable_executor.py in submit(self, fn, *args, **kwargs)
    175     def submit(self, fn, *args, **kwargs):
    176         with self._submit_resize_lock:
--> 177             return super(_ReusablePoolExecutor, self).submit(
    178                 fn, *args, **kwargs)
    179

c:\Users\kathe\OneDrive\Documents\Anaconda\lib\site-packages\joblib\externals\loky\pr
ocess_executor.py in submit(self, fn, *args, **kwargs)
   1113         with self._flags.shutdown_lock:
   1114             if self._flags.broken is not None:
-> 1115                 raise self._flags.broken
   1116             if self._flags.shutdown:
   1117                 raise ShutdownExecutorError(

BrokenProcessPool: A task has failed to un-serialize. Please ensure that the argument
s of the function are all picklable.
```

In [52]:

```
# Predict on validation and test sets using the tuned model
y_val_pred = best_rf.predict(X_val_scaled)
y_test_pred = best_rf.predict(X_test_scaled)
```

In [53]:

```
# Validation set evaluation
val_mse = mean_squared_error(y_val, y_val_pred)
val_r2 = r2_score(y_val, y_val_pred)
```

```
print(f"Tuned Model Validation MSE: {val_mse}")
print(f"Tuned Model Validation R²: {val_r2}")

# Test set evaluation
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f"Tuned Model Test MSE: {test_mse}")
print(f"Tuned Model Test R²: {test_r2}")
```

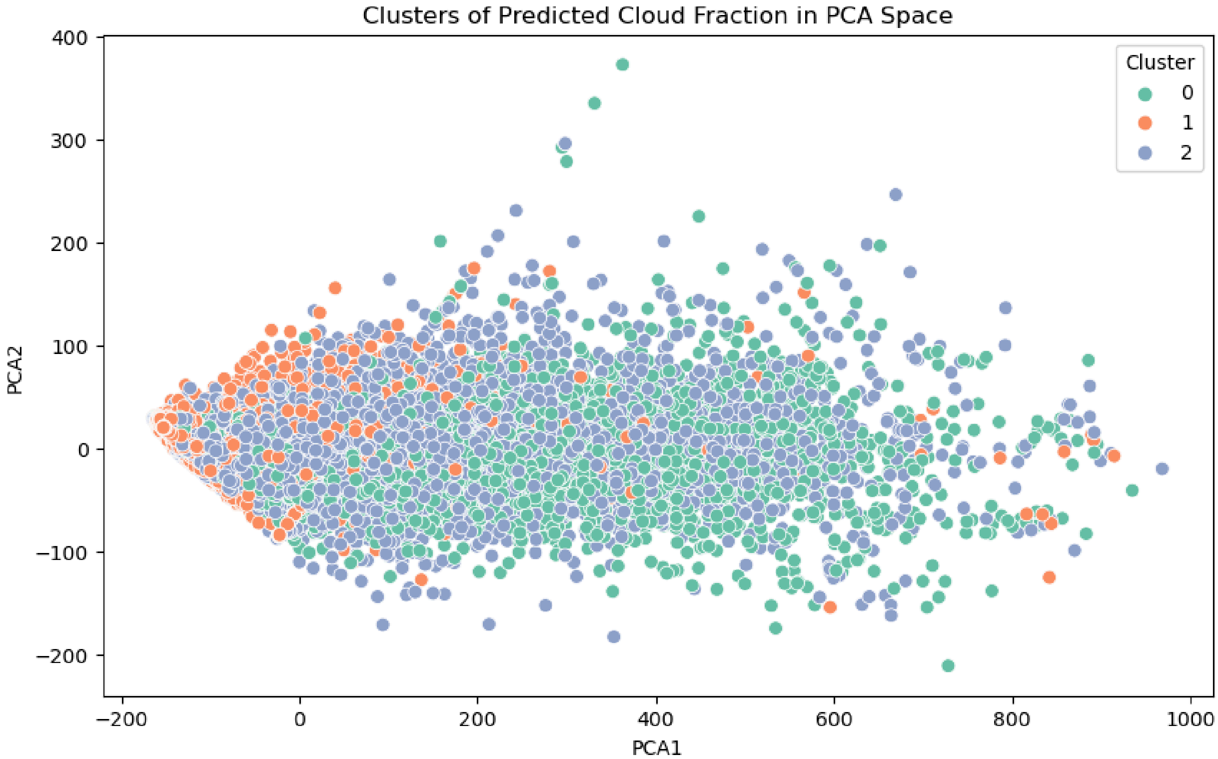
```
Tuned Model Validation MSE: 0.07807762430557745
Tuned Model Validation R²: 0.15308074124896198
Tuned Model Test MSE: 0.11331756689663158
Tuned Model Test R²: -0.041789268282844017
```

Since the metrics are worse, the hyperparameter tuning over complicated the model. Do not use this.

We need to try clustering again, but this time after we've already trained the model.

In [133…

```
from sklearn.cluster import KMeans


# Step 1: Obtain predictions from the original Random Forest model
# Assuming `y_test_pred` and `X_test` are already available
predicted_results = pd.DataFrame({'PredictedCloudFraction': y_test_pred})
predicted_results = pd.concat([predicted_results, X_test.reset_index(drop=True)], axis

# Step 2: Apply KMeans clustering
n_clusters = 3  # You can adjust this based on your data
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
predicted_results['Cluster'] = kmeans.fit_predict(predicted_results[['PredictedCloudFr

# Step 3: Visualize clusters
pca = PCA(n_components=2)
pca_results = pca.fit_transform(predicted_results[['PredictedCloudFraction'] + list(X_
predicted_results['PCA1'] = pca_results[:, 0]
predicted_results['PCA2'] = pca_results[:, 1]

plt.figure(figsize=(10, 6))
sns.scatterplot(data=predicted_results, x='PCA1', y='PCA2', hue='Cluster', palette='Se
plt.title('Clusters of Predicted Cloud Fraction in PCA Space')
plt.show()

# Step 4: Analyze clusters
cluster_summary = predicted_results.groupby('Cluster').mean()
print(cluster_summary)
```



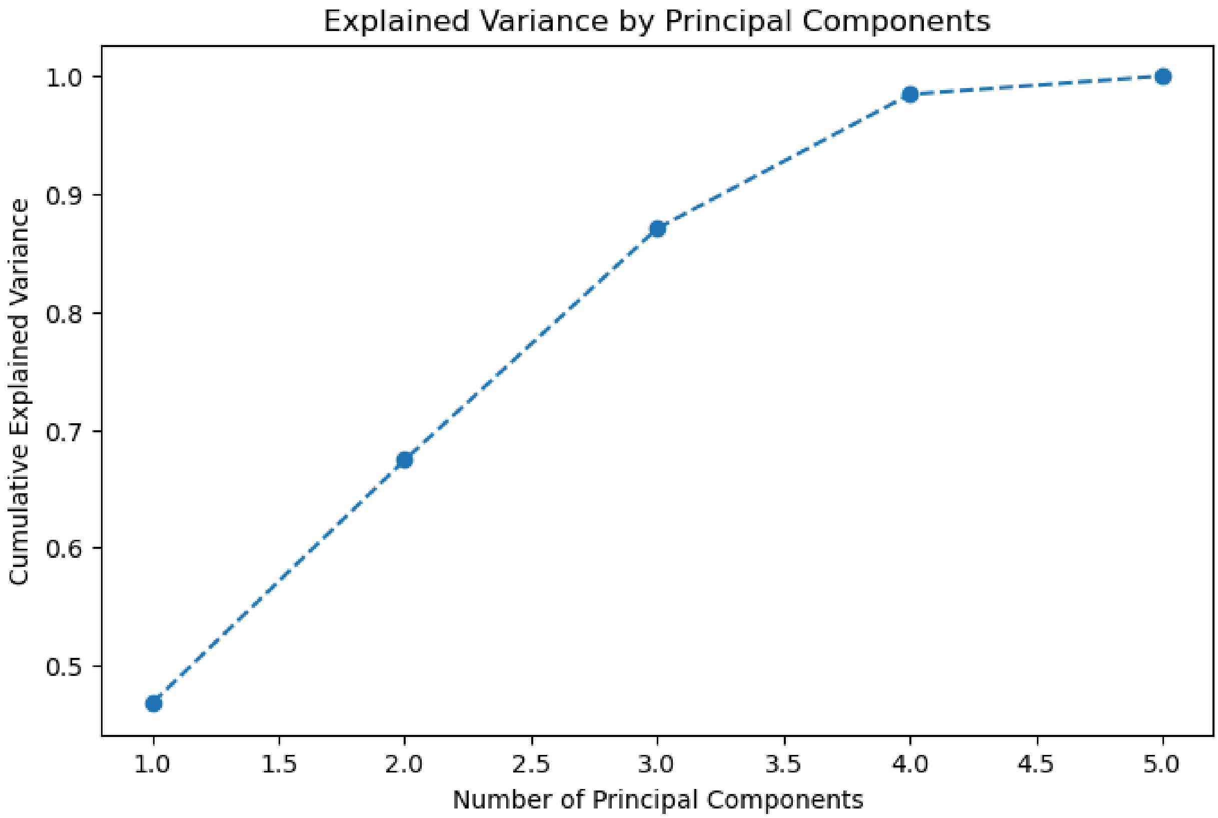

```
         PredictedCloudFraction  ReflectedSW  CloudOpticalDepth  \
Cluster
0                      0.592181   216.405666           6.891280
1                      0.219748   121.841765           7.411391
2                      0.398940   176.930672           6.608207

         CloudTopTemperature  DownwardSW  TotalPrecipWater       PCA1  \
Cluster
0                 249.861400  849.979963         27.274183  79.903119
1                 257.006609  982.156917         25.761045 -82.086715
2                 258.719928  904.982606         27.408686  12.408251

             PCA2
Cluster
0       -6.684390
1        6.600862
2       -0.857122
```

Principal component analysis can reduce dimensionality. We can add this in and then retrain and run a linear regression model to see if it improves.

In [134…

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Perform PCA on the scaled feature set
pca = PCA()
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Plot explained variance ratio to choose the number of components
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_rati
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
```

```
plt.title('Explained Variance by Principal Components')
plt.show()
```



In [135…
```
n_components = 3
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

In [136…
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Train the model
lr = LinearRegression()
lr.fit(X_train_pca, y_train)

# Predict on the test set
y_test_pca_pred = lr.predict(X_test_pca)

# Evaluate performance
test_mse_pca = mean_squared_error(y_test, y_test_pca_pred)
test_r2_pca = r2_score(y_test, y_test_pca_pred)

print(f"Linear Regression (PCA) Test MSE: {test_mse_pca}")
print(f"Linear Regression (PCA) Test R²: {test_r2_pca}")
```

```
Linear Regression (PCA) Test MSE: 0.10978557760019332
Linear Regression (PCA) Test R²: -0.009317793246005568
```

In [137…
```
components = pd.DataFrame(pca.components_, columns=feature_columns)
print(components)
```

```
   ReflectedSW  CloudOpticalDepth  CloudTopTemperature  DownwardSW  \
0     0.617879           0.484732            -0.001885   -0.617139
1     0.030721          -0.181080             0.762663   -0.064883
2    -0.043309           0.004087            -0.625187    0.023536

   TotalPrecipWater
0          0.048947
1          0.616768
2          0.778906
```

Try the PCA on the RandomForest.

In [138…

```python
# Apply PCA to the feature set
pca = PCA(n_components=3)  # Use 3 components as decided earlier
X_train_pca = pca.fit_transform(X_train_scaled)
X_val_pca = pca.transform(X_val_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train a Random Forest on the PCA-transformed features
rf_pca = RandomForestRegressor(random_state=42, n_estimators=100, max_depth=10)
rf_pca.fit(X_train_pca, y_train)

# Make predictions
y_val_pred_pca = rf_pca.predict(X_val_pca)
y_test_pred_pca = rf_pca.predict(X_test_pca)

# Evaluate performance
val_mse_pca = mean_squared_error(y_val, y_val_pred_pca)
val_r2_pca = r2_score(y_val, y_val_pred_pca)
test_mse_pca = mean_squared_error(y_test, y_test_pred_pca)
test_r2_pca = r2_score(y_test, y_test_pred_pca)

print(f"PCA + Random Forest Validation MSE: {val_mse_pca}")
print(f"PCA + Random Forest Validation R²: {val_r2_pca}")
print(f"PCA + Random Forest Test MSE: {test_mse_pca}")
print(f"PCA + Random Forest Test R²: {test_r2_pca}")
```

```
PCA + Random Forest Validation MSE: 0.0810710459929837
PCA + Random Forest Validation R²: 0.12061066420479805
PCA + Random Forest Test MSE: 0.11337223992808555
PCA + Random Forest Test R²: -0.042291906832127246
```

RandomForest with PCA is slightly worse than RandomForest without PCA.

Use a neural network instead of random forest or linear regression.