ET0_NN

May 14, 2025

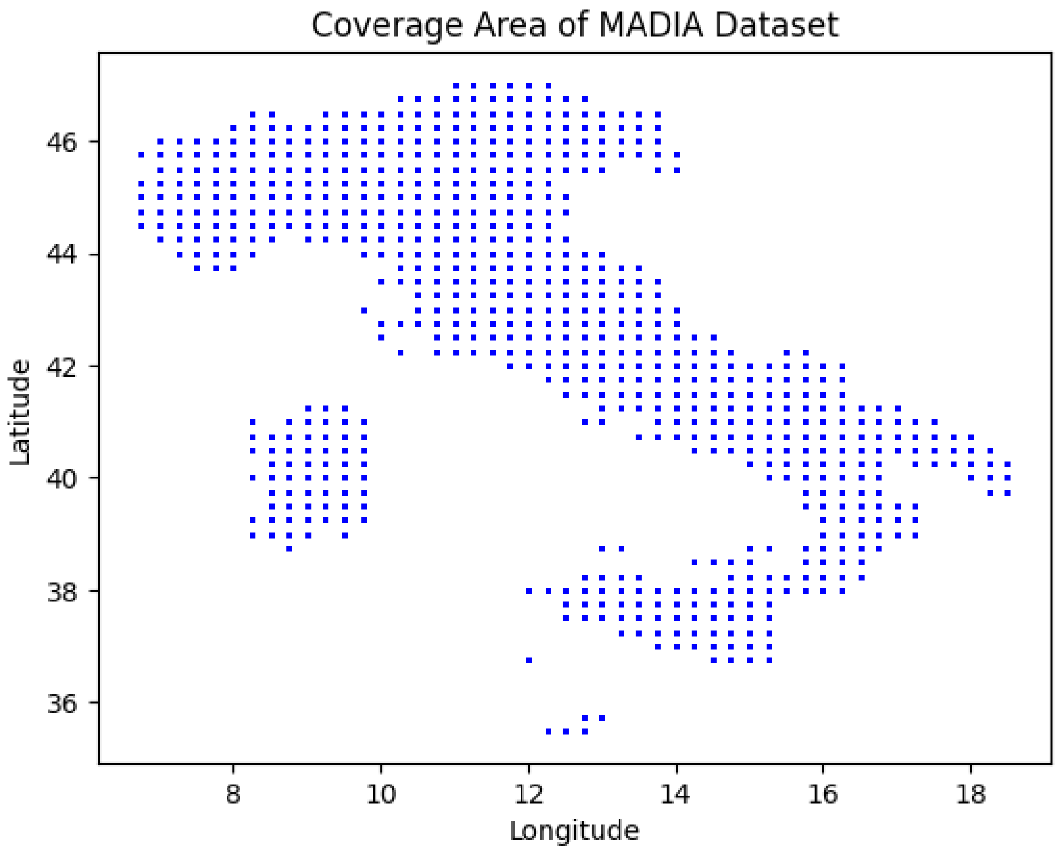
[63]:
```python
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from datetime import datetime
import pandas as pd
from glob import glob
```

[64]:
```python
#** load the dataframe of climatological variables **

# load data from the csv files
csv_files = sorted(glob("./MADIA_daily_dataset_v1.3/csv_data/*_e5_d.csv"))
csv_files = [file for file in csv_files if 1991 <= int(file.split('/')[-1].
 ↪split('_')[0]) <= 1992]

dailyClimatological_data = pd.DataFrame()

for file in csv_files:
    df = pd.read_csv(file)

    dailyClimatological_data = pd.concat([dailyClimatological_data, df],␣
 ↪ignore_index=True)

print(dailyClimatological_data.head())

# tasmin    mean of daily minimum near-surface air temperature
# tasmean   mean of daily average near-surface air temperature
# tasmax    mean of daily maximum near-surface air temperature
# rhmin     mean of daily minimum near-surface relative air humidity
# rhmax     mean of daily maximum near-surface relative air humidity
# ws10      mean of daily wind speed
# ssrd      mean of daily surface solar radiation downwards (shortwave␣
 ↪radiation)
# ppn       sum of daily depth of water-equivalent precipitation
# pev       sum of daily crop reference evapotranspiration estimated by FAO␣
 ↪Penman-Monteith method
```

```
# zg        geopotential height: average cell height (metres) above the geoid,␣
↪which corresponds approximately to the elevation
# dekad     number of dekad from the beginning of the year
# expver    code which identi es temporary data when expver=5
# mask      boolean code to identify cells belonging to the Italian country
```

```
   longitude  latitude        time     tasmin     tasmax    tasmean     rhmin  \
0      12.25      35.5  1991-01-01  14.007111  15.914581  14.960846  0.800915
1      12.25      35.5  1991-01-02  14.251373  15.310181  14.780777  0.657868
2      12.25      35.5  1991-01-03  14.284302  14.663757  14.474030  0.655431
3      12.25      35.5  1991-01-04  14.027557  15.211334  14.619446  0.676937
4      12.25      35.5  1991-01-05  13.894623  15.612396  14.753509  0.693268

      rhmax       ws10       ssrd       ppn       pev  expver  mask
0  0.893762   5.854601   9.278107  0.018810  1.418778     1.0   1.0
1  0.851364  10.327004  10.670974  0.036794  2.340696     1.0   1.0
2  0.674182   5.341772   8.324915  0.000000  2.223704     1.0   1.0
3  0.854900   5.503022   9.915781  0.008129  1.806671     1.0   1.0
4  0.839577   7.949931   8.691664  0.330742  2.051346     1.0   1.0
```

[65]:
```
# **show a plot of the coverage area of the MADIA dataset**

plt.scatter(dailyClimatological_data['longitude'],␣
↪dailyClimatological_data['latitude'], s=1, c='blue', alpha=0.5)
plt.title('Coverage Area of MADIA Dataset')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

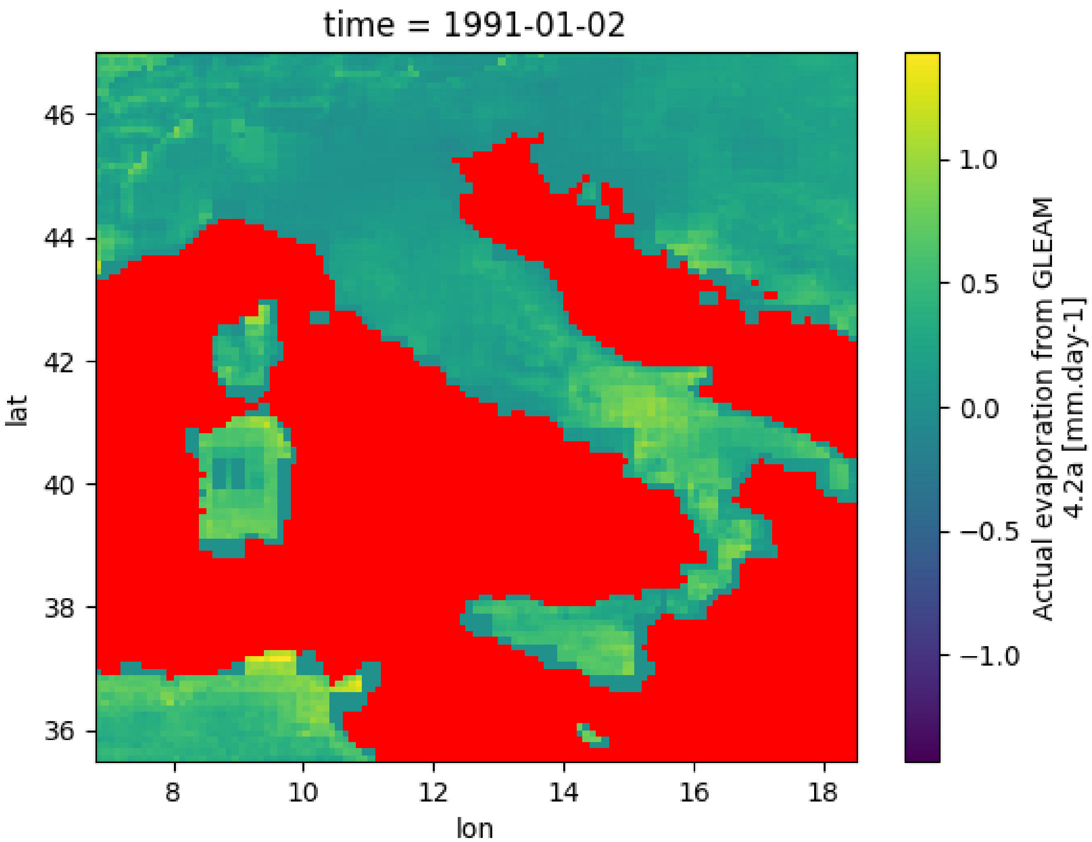



```
[66]: #** load the dataset of Evapotranspiration from the nc file
nc_files = sorted(glob("./GLEM_daily_dataset/E_*_GLEAM_v4.2a.nc"))

dailyE_data = []

for file in nc_files:
    ds = xr.open_dataset(file)

    # the evapotranspiration data are referenced to the Global territory
    E_lon = ds['lon'].values
    E_lat = ds['lat'].values

    # the climatological data are referenced to the Italian territory
    clim_lon = dailyClimatological_data['longitude'].values
    clim_lat = dailyClimatological_data['latitude'].values

    # select the evapotranspiration data for the Italian territory
    E_lon_idx = np.where((E_lon >= clim_lon.min()) & (E_lon <= clim_lon.max()))
    E_lat_idx = np.where((E_lat >= clim_lat.min()) & (E_lat <= clim_lat.max()))
```

```
    E_lon = E_lon[E_lon_idx]
    E_lat = E_lat[E_lat_idx]

    E_data = ds['E'].sel(lon=E_lon, lat=E_lat)
    dailyE_data.append(E_data)

dailyE_data = xr.concat(dailyE_data, dim='time')
```

[67]:
```
#** show a plot of the Evapotranspiration data **
et = dailyE_data.isel(time=1)

cmap = cm.get_cmap('viridis').copy()    # create a copy of the colormap
cmap.set_bad(color='red')  # highlight NaN values in red

et.plot(cmap=cmap)  # plot the data with the modified colormap

plt.show()

print(f"Daily Evapotranspiration shape (time, latitude, longitude):␣
 ↪{dailyE_data.shape}")

# count the number of pixel that are NaN at any time
nan_pixels = np.isnan(dailyE_data).any(dim='time').sum().item()
print(f"Number of NaN values in the dataset: {nan_pixels}")
```

```
/tmp/ipykernel_25375/3907745109.py:4: MatplotlibDeprecationWarning: The get_cmap
function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use
``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or
``pyplot.get_cmap()`` instead.
  cmap = cm.get_cmap('viridis').copy()    # create a copy of the colormap
```




```
Daily Evapotranspiration shape (time, latitude, longitude): (731, 115, 117)
Number of NaN values in the dataset: 6854
```

[68]:

```
#** evaluate the elapsed_days **
dailyClimatological_data['time'] = pd.
 ↪to_datetime(dailyClimatological_data['time'])
dailyClimatological_data['elapsed_days'] = (dailyClimatological_data['time'] -␣
 ↪dailyClimatological_data['time'].min()).dt.days.values

print(dailyClimatological_data.head())
print(dailyClimatological_data.tail())
```

```
   longitude  latitude       time     tasmin     tasmax   tasmean     rhmin  \
0      12.25      35.5 1991-01-01  14.007111  15.914581  14.960846  0.800915
1      12.25      35.5 1991-01-02  14.251373  15.310181  14.780777  0.657868
2      12.25      35.5 1991-01-03  14.284302  14.663757  14.474030  0.655431
3      12.25      35.5 1991-01-04  14.027557  15.211334  14.619446  0.676937
4      12.25      35.5 1991-01-05  13.894623  15.612396  14.753509  0.693268

      rhmax      ws10      ssrd       ppn       pev  expver  mask  \
0  0.893762  5.854601  9.278107  0.018810  1.418778     1.0   1.0
```

```
1  0.851364  10.327004  10.670974  0.036794  2.340696     1.0   1.0
2  0.674182   5.341772   8.324915  0.000000  2.223704     1.0   1.0
3  0.854900   5.503022   9.915781  0.008129  1.806671     1.0   1.0
4  0.839577   7.949931   8.691664  0.330742  2.051346     1.0   1.0

   elapsed_days
0             0
1             1
2             2
3             3
4             4
        longitude  latitude       time     tasmin     tasmax    tasmean  \
533625      12.25      47.0 1992-12-27 -16.907928  -8.750977 -12.829453
533626      12.25      47.0 1992-12-28 -20.676544 -13.855988 -17.266266
533627      12.25      47.0 1992-12-29 -18.643692  -9.086395 -13.865044
533628      12.25      47.0 1992-12-30 -12.737884  -8.615723 -10.676804
533629      12.25      47.0 1992-12-31 -11.646362  -8.826569 -10.236465

           rhmin     rhmax      ws10      ssrd       ppn       pev  expver  \
533625  0.477988  0.899087  1.686623  6.241381  0.016790  0.199788     1.0
533626  0.503882  0.888908  1.537032  6.182924  0.074096  0.142260     1.0
533627  0.386627  0.598119  0.879861  6.261357  0.000000  0.154225     1.0
533628  0.475457  0.602618  0.837980  6.335670  0.000000  0.130916     1.0
533629  0.475907  0.592939  1.301756  6.304410  0.000000  0.217792     1.0

        mask  elapsed_days
533625   1.0           726
533626   1.0           727
533627   1.0           728
533628   1.0           729
533629   1.0           730
```

[69]:

```
#** associate the Evapotranspiration data to the climatological data **

# build a 2D grid of lat/lon points with the evapotranspiration data
grid_lat = dailyE_data['lat'].values
grid_lon = dailyE_data['lon'].values
grid_lon2d, grid_lat2d = np.meshgrid(grid_lon, grid_lat)

from scipy.spatial import cKDTree

# build a KDTree for fast nearest-neighbor search
grid_points = np.column_stack([grid_lat2d.ravel(), grid_lon2d.ravel()])
tree = cKDTree(grid_points)

# extract (lat, lon) from the climate data
query_points = dailyClimatological_data[['latitude', 'longitude']].values
```

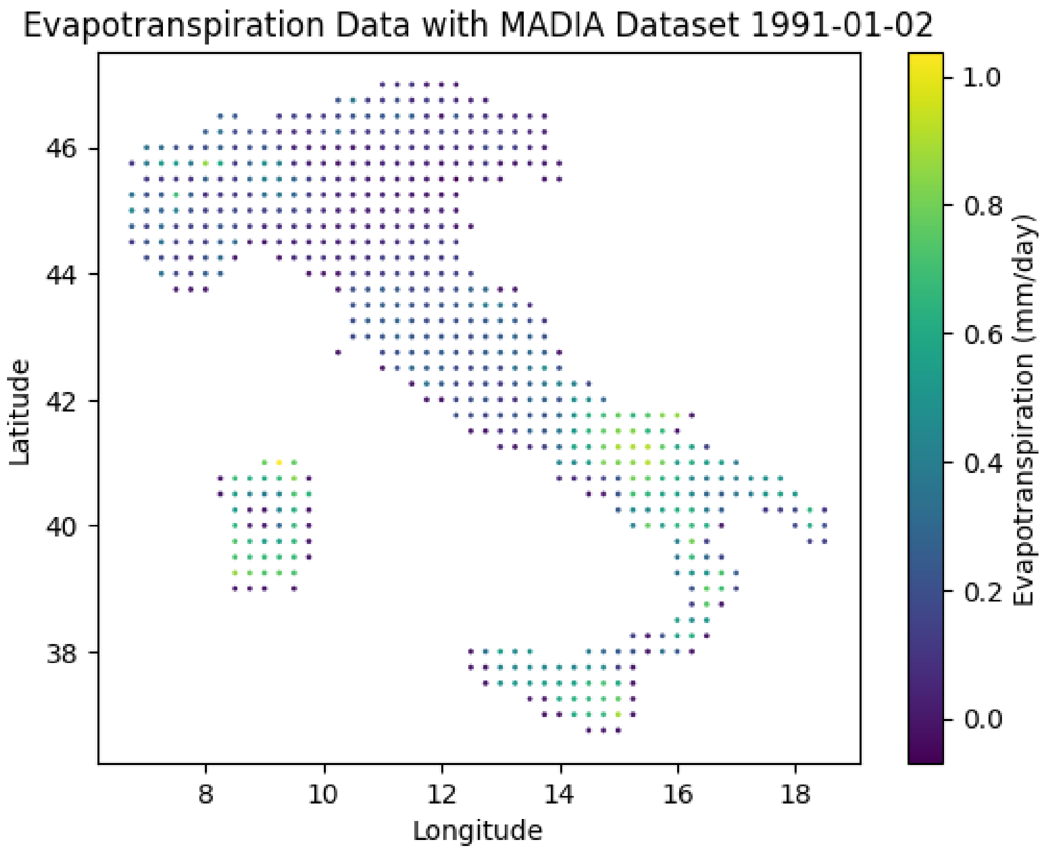
```python
# query the nearest grid points once
_, indices = tree.query(query_points)

# convert flat indices to 2D (lat_idx, lon_idx)
lat_idx, lon_idx = np.unravel_index(indices, grid_lat2d.shape)

# time indices, evaluate the amount of days elapsed since the first day of the␣
↪dataset
time_idx = dailyClimatological_data['elapsed_days']

# get values from xarray using vectorized indexing
E_values = dailyE_data.isel(
    time=xr.DataArray(time_idx, dims='points'),
    lat=xr.DataArray(lat_idx, dims='points'),
    lon=xr.DataArray(lon_idx, dims='points')
).values

# Assign to DataFrame
dailyClimatological_data['E'] = E_values

print(dailyClimatological_data.head())
print(dailyClimatological_data.tail())
```

```
   longitude  latitude       time     tasmin     tasmax    tasmean     rhmin  \
0      12.25      35.5 1991-01-01  14.007111  15.914581  14.960846  0.800915
1      12.25      35.5 1991-01-02  14.251373  15.310181  14.780777  0.657868
2      12.25      35.5 1991-01-03  14.284302  14.663757  14.474030  0.655431
3      12.25      35.5 1991-01-04  14.027557  15.211334  14.619446  0.676937
4      12.25      35.5 1991-01-05  13.894623  15.612396  14.753509  0.693268

      rhmax       ws10       ssrd       ppn       pev  expver  mask  \
0  0.893762   5.854601   9.278107  0.018810  1.418778     1.0   1.0
1  0.851364  10.327004  10.670974  0.036794  2.340696     1.0   1.0
2  0.674182   5.341772   8.324915  0.000000  2.223704     1.0   1.0
3  0.854900   5.503022   9.915781  0.008129  1.806671     1.0   1.0
4  0.839577   7.949931   8.691664  0.330742  2.051346     1.0   1.0

   elapsed_days   E
0             0 NaN
1             1 NaN
2             2 NaN
3             3 NaN
4             4 NaN
        longitude  latitude       time     tasmin     tasmax    tasmean  \
533625      12.25      47.0 1992-12-27 -16.907928  -8.750977 -12.829453
533626      12.25      47.0 1992-12-28 -20.676544 -13.855988 -17.266266
533627      12.25      47.0 1992-12-29 -18.643692  -9.086395 -13.865044
```

```
533628       12.25       47.0 1992-12-30 -12.737884  -8.615723 -10.676804
533629       12.25       47.0 1992-12-31 -11.646362  -8.826569 -10.236465

           rhmin     rhmax      ws10      ssrd       ppn       pev  expver  \
533625  0.477988  0.899087  1.686623  6.241381  0.016790  0.199788     1.0
533626  0.503882  0.888908  1.537032  6.182924  0.074096  0.142260     1.0
533627  0.386627  0.598119  0.879861  6.261357  0.000000  0.154225     1.0
533628  0.475457  0.602618  0.837980  6.335670  0.000000  0.130916     1.0
533629  0.475907  0.592939  1.301756  6.304410  0.000000  0.217792     1.0

        mask  elapsed_days         E
533625   1.0           726  0.034045
533626   1.0           727  0.021541
533627   1.0           728  0.055241
533628   1.0           729  0.025559
533629   1.0           730  0.024122
```

[70]:

```
#** show a plot of the Evapotranspiration data with the MADIA dataset **
et = dailyClimatological_data[dailyClimatological_data['elapsed_days'] == 1]
plt.scatter(et['longitude'], et['latitude'], s=1, c=et['E'], cmap='viridis')
plt.colorbar(label='Evapotranspiration (mm/day)')
plt.title(f'Evapotranspiration Data with MADIA Dataset {et["time"].dt.
 ↪strftime("%Y-%m-%d").values[0]}')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



[71]:
```
#** drop the rows with NaN values in the Evapotranspiration column **
print(f"Number of rows before dropping NaN values:␣
 ↪{len(dailyClimatological_data)}")
dailyClimatological_data = dailyClimatological_data.dropna(subset=['E'])
print(f"Number of rows after dropping NaN values:␣
 ↪{len(dailyClimatological_data)}")
```

```
Number of rows before dropping NaN values: 533630
Number of rows after dropping NaN values: 464185
```

[72]:
```
#** print the fields of the dataframe **
print(dailyClimatological_data.columns)
```

```
Index(['longitude', 'latitude', 'time', 'tasmin', 'tasmax', 'tasmean', 'rhmin',
       'rhmax', 'ws10', 'ssrd', 'ppn', 'pev', 'expver', 'mask', 'elapsed_days',
       'E'],
      dtype='object')
```

[73]:
```
#** define the features and the target variable **

# features (X) - all meteorological variables and time
```

```
# should i use time (the date of the day) or the elapsed days (amount of days␣
↪since the first day of the dataset) as a feature?
# should i use the ETo (potential evapotranspiration) estimated with the␣
↪Penman-Monteith equation as a feature?
X = dailyClimatological_data[['longitude', 'latitude', 'elapsed_days',␣
↪'tasmin', 'tasmax', 'tasmean', 'rhmin','rhmax', 'ws10', 'ssrd', 'ppn',␣
↪'pev']]

print(f"Feature matrix:\n{X.columns}")

# target (y) - ET (evapotranspiration)
y = dailyClimatological_data[['E']]

print(f"Target vector:\n{y.columns}")
```

```
Feature matrix:
Index(['longitude', 'latitude', 'elapsed_days', 'tasmin', 'tasmax', 'tasmean',
       'rhmin', 'rhmax', 'ws10', 'ssrd', 'ppn', 'pev'],
      dtype='object')
Target vector:
Index(['E'], dtype='object')
```

[74]:

```
from sklearn.preprocessing import StandardScaler

# initialize the scaler
scaler = StandardScaler()

# fit the scaler on the feature set and transform the features
X_scaled = scaler.fit_transform(X)
```

[ ]:

```
# X_scaled should already be in the correct shape after scaling
# If X is still in a DataFrame, convert it to a numpy array
X_scaled = X_scaled.reshape((X_scaled.shape[0], X_scaled.shape[1]))  #␣
↪(samples, features)
```

[76]:

```
# from sklearn.model_selection import train_test_split

# # Split the data into train and test sets (80/20 split)
# X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪2, random_state=42)
```

[77]:

```
# import tensorflow as tf
# from tensorflow.keras import layers, models

# # Build the NN model
# model = models.Sequential()
```

```
# # Input layer
# model.add(layers.Dense(64, input_dim=X_train.shape[1], activation='relu'))

# # Hidden layers
# model.add(layers.Dense(64, activation='relu'))
# model.add(layers.Dense(32, activation='relu'))

# # Output layer (single neuron for regression)
# model.add(layers.Dense(1))

# # Compile the model
# model.compile(optimizer='adam', loss='mean_squared_error')

# # Train the model
# history = model.fit(X_train, y_train, epochs=50, batch_size=32,␣
↪validation_split=0.2)
```

[78]:
```
# # Evaluate the model on the test set
# loss = model.evaluate(X_test, y_test)

# # Print the test loss
# print(f"Test loss: {loss}")

# # If you want to also predict the values on the test set
# y_pred = model.predict(X_test)
```

[79]:
```
# import matplotlib.pyplot as plt

# # Plot the loss curve
# plt.plot(history.history['loss'], label='Train Loss')
# plt.plot(history.history['val_loss'], label='Validation Loss')
# plt.title('Loss over epochs')
# plt.xlabel('Epochs')
# plt.ylabel('Loss')
# plt.legend()
# plt.show()

# # Scatter plot of actual vs predicted ET
# plt.scatter(y_test, y_pred)
# plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
↪lw=2)  # Identity line
# plt.xlabel('True ET')
# plt.ylabel('Predicted ET')
# plt.title('True vs Predicted Evapotranspiration')
# plt.show()
```

[92]:
```
n_days = dailyClimatological_data['elapsed_days'].nunique()
n_features = 10
lat_vals = np.sort(dailyClimatological_data['latitude'].unique())   # height
lon_vals = np.sort(dailyClimatological_data['longitude'].unique())  # width
height = len(lat_vals)
width = len(lon_vals)

# initialize tensor with NaN values and shape (n_days, n_features, height,␣
↪width)
data_tensor = np.full((n_days, n_features, height, width), np.nan, dtype=np.
↪float32)

# map latitude, longitude and time values to indices
lat_map = {val: idx for idx, val in enumerate(lat_vals)}
lon_map = {val: idx for idx, val in enumerate(lon_vals)}
day_map = {val: idx for idx, val in enumerate(np.
↪sort(dailyClimatological_data['elapsed_days'].unique()))}

# fill each pixel of the tensor with the corresponding value
for _, row in dailyClimatological_data.iterrows():
    day_idx = day_map[row['elapsed_days']]
    y_idx = lat_map[row['latitude']]      # y/height
    x_idx = lon_map[row['longitude']]     # x/width

    features = [
        row['tasmin'], row['tasmax'], row['tasmean'],
        row['rhmin'], row['rhmax'], row['ws10'],
        row['ssrd'], row['ppn'], row['pev'], row['E']
    ]

    data_tensor[day_idx, :, y_idx, x_idx] = features
```

[96]:
```
# extract features and target variable
X = data_tensor[:, :-1, :, :]  # shape: (days, 9, H, W)
print(f"Feature tensor shape: {X.shape}")
y = data_tensor[:, -1, :, :]   # shape: (days, H, W) - actual ET
print(f"Target tensor shape: {y.shape}")
```

```
Feature tensor shape: (731, 9, 42, 48)
Target tensor shape: (731, 42, 48)
```

[ ]:
```
```

```
Feature tensor shape: (731, 42, 48, 9)
n_days: 731, height: 42, width: 48, n_features: 9
Epoch 1/50
```

```
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
Cell In[102], line 28
     26 print(f"Feature tensor shape: {X.shape}")
     27 print(f"n_days: {n_days}, height: {height}, width: {width}, n_features:␣
 ↪{n_features-1}")
---> 28 history = model.fit(X, y, epochs=50, batch_size=32, validation_split=0.2)

File ~/Backup/Magistrale/InovativeWirelessPlatform4IoT/.venv/lib/python3.10/
 ↪site-packages/keras/src/utils/traceback_utils.py:122, in filter_traceback.
 ↪<locals>.error_handler(*args, **kwargs)
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

File ~/Backup/Magistrale/InovativeWirelessPlatform4IoT/.venv/lib/python3.10/
 ↪site-packages/keras/src/models/functional.py:273, in Functional.
 ↪_adjust_input_rank(self, flat_inputs)
    271             adjusted.append(ops.expand_dims(x, axis=-1))
    272             continue
--> 273     raise ValueError(
    274         f"Invalid input shape for input {x}. Expected shape "
    275         f"{ref_shape}, but input has incompatible shape {x.shape}"
    276     )
    277 # Add back metadata.
    278 for i in range(len(flat_inputs)):

ValueError: Exception encountered when calling Sequential.call().

Invalid input shape for input Tensor("data:0", shape=(None, 42, 48, 9),␣
 ↪dtype=float32). Expected shape (None, 731, 42, 48, 9), but input has␣
 ↪incompatible shape (None, 42, 48, 9)

Arguments received by Sequential.call():
  • inputs=tf.Tensor(shape=(None, 42, 48, 9), dtype=float32)
  • training=True
  • mask=None
```

[ ]:
```python
# Evaluate the model on the test set
loss = model.evaluate(X, y)

# Print the test loss
print(f"Test loss: {loss}")

# Predict the values on the test set
```

```
y_pred = model.predict(X)

# Reshape the predictions to match the original data shape
y_pred_reshaped = y_pred.reshape((y_pred.shape[0], height, width))
```

[ ]:
```
# Plot the first day of predictions
plt.imshow(y_pred_reshaped[0], cmap='viridis')
plt.colorbar(label='Evapotranspiration (mm/day)')
plt.title('Predicted Evapotranspiration for Day 1')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

# Scatter plot of actual vs predicted ET
plt.scatter(y.flatten(), y_pred.flatten())
plt.plot([min(y.flatten()), max(y.flatten())], [min(y.flatten()), max(y.
 ↪flatten())], color='red', lw=2)  # Identity line
plt.xlabel('True ET')
plt.ylabel('Predicted ET')
plt.title('True vs Predicted Evapotranspiration')
plt.show()

# Save the model
model.save('et_model.h5')
```