```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import netCDF4 as nc
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import xarray as xr
import cartopy.crs as ccrs
import PIL
import imageio
import pandas as pd
from matplotlib.animation import FuncAnimation
from netCDF4 import Dataset


# In[2]:


data = xr.open_dataset('D:/data.nc')
t2m_ = data['t2m'] - 273.15
d2m_ = data['d2m'] - 273.15
u10_ = data['u10']
v10_ = data['v10']
times_ = data['time']
times = data['time']
tp = pre['tp']
pre = xr.open_dataset('D:/pre.nc')
data3 = xr.open_dataset('D:/data3.nc')
data4 = xr.open_dataset('D:/data4.nc')
data_cape = xr.open_dataset('D:/cape.nc')
dataset = xr.open_dataset('D:/data6.nc')
RH = xr.open_dataarray('D:/data2.nc')   # Relative humidity from another file
SH = xr.open_dataarray('D:/data3.nc')   # Specific humidity from another file
SP = xr.open_dataarray('D:/data4.nc')    # Surface Pressure from another file

png_dir = 'D:'

latitude = data3['latitude']
longitude = data3['longitude']

file = 'D:/data.nc'
data = Dataset(file, mode='r')

# Selecting the variable 'z' from the dataset
height_data = dataset['z']
```

```
# In[3]:


# Spatial information
lats = data.variables['latitude'][:]
longs = data.variables['longitude'][:]
lat = data.variables['latitude'][:]
lon = data.variables['longitude'][:]

# Temporal information
time = data.variables['time'][:]

t2m = data.variables['t2m'][:] # The 2-meter temperature in Kelvin
t2m = t2m - 273.15 # Convert to Celsius

d2m = data.variables['d2m'][:] # The 2-meter dewpoint temperature in Kelvin
d2m = d2m - 273.15 # Convert to Celsius

u10 = data.variables['u10'][:] # The 10-meter U wind component in m/s
v10 = data.variables['v10'][:] # The 10-meter V wind component in m/s


# In[66]:


# Time-Series plotting of Temperature
t2m_adiyaman = t2m_.sel(latitude=37.76441,longitude=38.27629,method='nearest')
# Adiyaman
t2m_adiyaman.plot.line(label='Adiyaman')
t2m_urfa =
t2m_.sel(latitude=   37.158333,longitude=38.791668,method='nearest') # Urfa
t2m_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Temperature (Hourly)")  # Setting the title as
"Hourly Temperature Time Series"
plt.xlabel("Date")  # Setting the x-axis label
plt.ylabel("Temperature [°C]")  # Setting the y-axis label

plt.xticks(rotation=45)  # Rotating the date labels on the x-axis
plt.tight_layout() # Adjusting layout to prevent cropping
plt.savefig(f'D:/output/t2m/TS/{1}.png')
plt.pause(1)
plt.show()


# In[67]:
```

```
# Time-Series plotting of DEW-POINT TEMPERATURE
d2m_adiyaman = d2m_.sel(latitude=37.76441,longitude=38.27629,method='nearest')
# Adiyaman
d2m_adiyaman.plot.line(label='Adiyaman')
d2m_urfa =
d2m_.sel(latitude=   37.158333,longitude=38.791668,method='nearest') # Urfa
d2m_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Dew-Point Temperature (Hourly)")
plt.xlabel("Date")
plt.ylabel("Dew-Point Temperature [°C]")

plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(f'D:/output/d2m/TS/{1}.png')
plt.pause(1)
plt.show()

# Time-Series plotting of Precipitation
pre_adiyaman = tp.sel(latitude=37.76441,longitude=38.27629,method='nearest') #
Adiyaman
pre_adiyaman.plot.line(label='Adiyaman')
pre_urfa = tp.sel(latitude= 37.158333,longitude=38.791668,method='nearest') #
Urfa
pre_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Total Precipitation (Hourly)")  # Setting the title as "Hourly
Temperature Time Series"
plt.xlabel("Date")  # Setting the x-axis label
plt.ylabel("Precipitation [mm]")  # Setting the y-axis label

plt.xticks(rotation=45)  # Rotating the date labels on the x-axis
plt.tight_layout() # Adjusting layout to prevent cropping
plt.savefig(f'D:/output/pre/{1}.png')
plt.pause(1)
plt.show()


# In[68]:


# # Time-Series plotting of Relative Humidity in %
RH_adiyaman = RH.sel(latitude=37.76441,longitude=38.27629,method='nearest') #
Adiyaman
RH_adiyaman.plot.line(label='Adiyaman')
```

```
RH_urfa = RH.sel(latitude=  37.158333,longitude=38.791668,method='nearest') #
Urfa
RH_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of RH (Hourly)")  # Set the title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Relative Humidity in [%]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout() # Adjust layout to prevent cropping
plt.savefig(f'D:/output/RH/TS/{1}.png')
plt.pause(1)
plt.show()


# In[69]:


#Calculation of MIXING RATIO by using the formula M = q / (1 - q)

q = data3.variables['q'][:] # Specific humidity in kg/kg
r = q / (1 - q) # Mixing ratio in kg/kg
r = r*1000 # Mixing ratio in g/kg


# In[71]:


# Time-Series plotting of Mixing Ratio in kg/kg for Adiyaman and Urfa
# Set latitude values as a coordinate
ds = xr.Dataset({'r': r}, coords={'latitude': lats, 'longitude': longs,
'time': times})

ds['latitude'] = lats

# Create a new 'Dataset' using the data set
ds_r = xr.Dataset({'r': r}, coords={'latitude': lats, 'longitude': longs,
'time': times})

# Select the desired locations
r_adiyaman = ds_r.sel(latitude=37.76441, longitude=38.27629, method='nearest')
r_urfa = ds_r.sel(latitude=37.158333, longitude=38.791668, method='nearest')

# Plot the time series
plt.plot(r_adiyaman.time, r_adiyaman.r, label='Adiyaman')
plt.plot(r_urfa.time, r_urfa.r, label='Urfa')

plt.legend()
```

```
plt.title("Time-Series of Calculated Mixing Ratio (Hourly)")  # Set the title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Mixing Ratio in [kg/kg]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout() # Adjust layout to prevent cropping

plt.savefig(f'D:\output\mixing ratio/{1}.png')
plt.pause(1)
plt.show()


# In[73]:


# Time-Series plotting
u10_adiyaman = u10_.sel(latitude=37.76441,longitude=38.27629,method='nearest')
# Adiyaman
u10_adiyaman.plot.line(label='Adiyaman')
u10_urfa =
u10_.sel(latitude=   37.158333,longitude=38.791668,method='nearest') # Urfa
u10_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Wind Speed in 'u' Direction (Hourly)")  # Set the
title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Wind Component [m/s]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout() # Adjust layout to prevent cropping
plt.savefig(f'D:/output/u10/TS/{1}.png')
plt.pause(1)
plt.show()


# In[74]:


# Time-Series plotting
v10_adiyaman = v10_.sel(latitude=37.76441,longitude=38.27629,method='nearest')
# Adiyaman
v10_adiyaman.plot.line(label='Adiyaman')
v10_urfa =
v10_.sel(latitude=   37.158333,longitude=38.791668,method='nearest') # Urfa
v10_urfa.plot.line(label='Urfa')

plt.legend()
```

```
plt.title("Time-Series of Wind Speed in 'v' Direction (Hourly)")  # Set the
title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Wind Component [m/s]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout() # Adjust layout to prevent cropping
plt.savefig(f'D:/output/v10/TS/{1}.png')
plt.pause(1)
plt.show()


# In[76]:


import numpy as np

# Calculate vector components of wind speed
wind_speed_adiyaman = np.sqrt(u10_adiyaman**2 + v10_adiyaman**2)
wind_speed_urfa = np.sqrt(u10_urfa**2 + v10_urfa**2)

# Plot u-component of wind speed
u10_adiyaman.plot.line(label='Adiyaman')
u10_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Wind Speed in 'u' Direction (Hourly)")  # Set the
title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Wind Component [m/s]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout()  # Adjust layout to prevent cropping
plt.savefig(f'D:/output/u10/TS/{1}.png')
plt.pause(1)
plt.show()

# Plot v-component of wind speed
v10_adiyaman.plot.line(label='Adiyaman')
v10_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Wind Speed in 'v' Direction (Hourly)")  # Set the
title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Wind Component [m/s]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout()  # Adjust layout to prevent cropping
```

```
plt.savefig(f'D:/output/v10/TS/{1}.png')
plt.pause(1)
plt.show()

# Plot vector components of wind speed
wind_speed_adiyaman.plot.line(label='Adiyaman')
wind_speed_urfa.plot.line(label='Urfa')

plt.legend()
plt.title("Time-Series of Wind Speed (Hourly)")  # Set the title
plt.xlabel("Date")  # Set the x-axis label
plt.ylabel("Wind Speed [m/s]")  # Set the y-axis label

plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.tight_layout()  # Adjust layout to prevent cropping
plt.savefig(f'D:/output/wind_speed/TS/{1}.png')
plt.pause(1)
plt.show()


# In[77]:


# Load the data for calculating the potential temperature
T = data.variables['t2m'][:]  # Temperature data (K)
p = data4.variables['sp'][:]  # ressure data (hPa)

# Setting the latitude/longitude values
lats = data.variables['latitude'][:]
lons = data.variables['longitude'][:]

# Setting the lat/lon values for Adıyaman and Şanlıurfa
lat_adiyaman = 37.76441
lon_adiyaman = 38.27629
lat_urfa = 37.158333
lon_urfa = 38.791668

# Finding the nearest lat/lon indices for Adıyaman and Şanlıurfa
lat_idx_adiyaman = np.abs(lats - lat_adiyaman).argmin()
lon_idx_adiyaman = np.abs(lons - lon_adiyaman).argmin()
lat_idx_urfa = np.abs(lats - lat_urfa).argmin()
lon_idx_urfa = np.abs(lons - lon_urfa).argmin()

# Calculation of potential temperature for Adıyaman
T_adiyaman = T[:, lat_idx_adiyaman, lon_idx_adiyaman]
p_adiyaman = p[:, lat_idx_adiyaman, lon_idx_adiyaman]
theta_adiyaman = T_adiyaman * (100000 / p_adiyaman) ** 0.286

# Calculation of potential temperature for Sanliurfa
```

```
T_urfa = T[:, lat_idx_urfa, lon_idx_urfa]
p_urfa = p[:, lat_idx_urfa, lon_idx_urfa]
theta_urfa = T_urfa * (100000 / p_urfa) ** 0.286

# Convert potential temperature values to DataFrame
df_adiyaman = pd.DataFrame({'Time': data.variables['time'][:], 'Theta':
theta_adiyaman})
df_urfa = pd.DataFrame({'Time': data.variables['time'][:], 'Theta':
theta_urfa})

# Setting a start date for converting time values
start_date = pd.to_datetime('2023-01-01')

# Converting time values and create a new column
df_adiyaman['Datetime'] = start_date + pd.to_timedelta(df_adiyaman['Time'],
unit='h')
df_urfa['Datetime'] = start_date + pd.to_timedelta(df_urfa['Time'], unit='h')

# Creating a new figure
fig, ax = plt.subplots()

# Setting the date format
date_fmt = mdates.DateFormatter('%m-%d')
ax.xaxis.set_major_formatter(date_fmt)

# Setting the day locator
day_locator = mdates.DayLocator()
ax.xaxis.set_major_locator(day_locator)

# Plotting the potential temperature values
ax.plot(df_adiyaman['Datetime'], df_adiyaman['Theta'], label='Adıyaman')
ax.plot(df_urfa['Datetime'], df_urfa['Theta'], label='Şanlıurfa',
color='#FF8C00')


# Setting the axis labels and title
ax.set_xlabel('Date - year2023')
ax.set_ylabel('Potential Temperature (K)')
ax.set_title('Potential Temperature in Adiyaman and Sanliurfa (Hourly)')

fig.autofmt_xdate()

plt.legend()
plt.savefig(f'D:\output\potential_temperature/{1}.png')
plt.pause(1)
plt.show()


# In[120]:
```

```
# Time-Series plotting
t2m_adiyaman = t2m_.sel(latitude=37.76441,longitude=38.27629,method='nearest')
# Adiyaman
t2m_urfa =
t2m_.sel(latitude=   37.158333,longitude=38.791668,method='nearest') # Urfa
t2m_adiyaman = t2m_adiyaman + 273.15
t2m_urfa = t2m_urfa + 273.15


# In[79]:


import netCDF4 as nc
import matplotlib.pyplot as plt
import numpy as np
import xarray as xr

# Open the dataset
dataset = xr.open_dataset('D:/data6.nc')

# Select the variable of interest (e.g., 'z' variable)
height_data = dataset['z']

# Specify the coordinates to retrieve the sounding data
longitude = 38.27629
latitude = 37.76441

# Retrieve the sounding data at the specified location
sounding_data = height_data.sel(longitude=longitude, latitude=latitude,
method='nearest')

# Get the pressure levels and height values
pressure_levels = sounding_data.level.values
height_values = sounding_data.values

# Transpose the arrays
pressure_levels = np.transpose(pressure_levels)
height_values = np.transpose(height_values)

# Filter the data above 400 hPa
pressure_levels_filtered = pressure_levels[pressure_levels > 400]
height_values_filtered = height_values[pressure_levels > 400]

# Plot the sounding data with pressure on the y-axis and height on the x-axis
plt.plot(height_values_filtered, pressure_levels_filtered, linewidth=0.1)
plt.gca().invert_yaxis()
plt.xlabel('Height (meters)')
```

```
plt.ylabel('Pressure (hPa)')
plt.title('Pressure Sounding Profile by Height (Above 400 hPa) in Adiyaman')
plt.grid(True)
plt.savefig(f'D:/output/height_in_a_sounding/{1}.png')
plt.show()


# In[80]:


# Specify the coordinates to retrieve the sounding data
longitude = 38.791668
latitude = 37.158333

# Retrieve the sounding data at the specified location
sounding_data = height_data.sel(longitude=longitude, latitude=latitude,
method='nearest')

# Get the pressure levels and height values
pressure_levels = sounding_data.level.values
height_values = sounding_data.values

# Transpose the arrays
pressure_levels = np.transpose(pressure_levels)
height_values = np.transpose(height_values)

# Filter the data above 400 hPa
pressure_levels_filtered = pressure_levels[pressure_levels > 400]
height_values_filtered = height_values[pressure_levels > 400]

# Plot the sounding data with pressure on the y-axis and height on the x-axis
plt.plot(height_values_filtered, pressure_levels_filtered, linewidth=0.1)
plt.gca().invert_yaxis()
plt.xlabel('Height (meters)')
plt.ylabel('Pressure (hPa)')
plt.title('Pressure Sounding Profile by Height (Above 400 hPa) in Sanliurfa')
plt.grid(True)
plt.savefig(f'D:/output/height_in_a_sounding/{2}.png')
plt.pause(1)
plt.show()


# In[82]:


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

```
cape = data_cape['cape'].sel(latitude=37.76441, longitude=38.27629,
method='nearest')
cape2 = data_cape['cape'].sel(latitude=37.158333, longitude=38.791668,
method='nearest')

# Get the time coordinates as pandas datetime objects
cape_dates = pd.to_datetime(cape['time'].values)
cape2_dates = pd.to_datetime(cape2['time'].values)

plt.plot(cape_dates, cape)
plt.plot(cape2_dates, cape2)
plt.xlabel('Date')
plt.ylabel('CAPE')
plt.title('CAPE Data')
plt.xticks(rotation=45)  # Rotate the date labels on the x-axis
plt.grid(True)
plt.savefig(f'D:/output/cape/loaded_data.png')
plt.show()


# In[84]:


import scipy.integrate as spi
import xarray as xr
import matplotlib.pyplot as plt

dataset_temp = xr.open_dataset('D:/temp_cape.nc')
dataset_rh = xr.open_dataset('D:/rh_cape2.nc')

# Step 1: Vertical profile of temperature and relative humidity
temperature_profile = dataset_temp['t'].sel(latitude=37.76441,
longitude=38.27629, method='nearest')
relative_humidity_profile = dataset_rh['r'].sel(latitude=37.76441,
longitude=38.27629, method='nearest')

temperature_profile_daily =
temperature_profile.resample(time='1D').mean(dim='time')
relative_humidity_profile_daily =
relative_humidity_profile.resample(time='1D').mean(dim='time')

# Step 2: Match temperature and relative humidity profiles with corresponding
levels
temperature_with_levels =
temperature_profile_daily.interp(level=dataset_temp['level'])
relative_humidity_with_levels =
relative_humidity_profile.interp(level=dataset_rh['level'])

# Step 2: Calculation of potential temperature profile
```

```
pressure_levels = dataset_temp['level'].values
potential_temperature_profile = temperature_profile_daily * (1000 /
pressure_levels) ** (2/7)

def calculate_cape(temperature, relative_humidity, pressure):
    # Calculate the virtual temperature
    virtual_temperature = temperature * (1 + 0.61 * relative_humidity / 100)

    # Calculate the temperature difference between the parcel and the
environment
    temperature_difference = virtual_temperature - temperature

    # Calculate the integrated value for CAPE
    integrated_value = temperature_difference * (9.8 / temperature) *
(pressure - 400)

    # Return the CAPE value
    return integrated_value

# Step 3: Moving the virtual air parcel upwards
cape_values = []  # List to store the CAPE values

parcel_pressure = 1000  # Initial pressure (e.g., 1000 hPa)
parcel_temperature = temperature_profile_daily.sel(level=parcel_pressure)
parcel_relative_humidity =
relative_humidity_profile.sel(level=parcel_pressure)
lapse_rate = -6.5  # Lapse rate in °C/km

while parcel_pressure >= 400:  # Move upward from 400 hPa
    parcel_pressure -= 100  # Move 100 hPa upward
    altitude_change = 100 * 0.1  # Approximate altitude change of 1 km for
every 100 hPa
    parcel_temperature += lapse_rate * altitude_change  # Update temperature
based on the lapse rate
    parcel_relative_humidity -= 10  # For example, decrease relative humidity
by 10% for every 100 hPa

    cape = calculate_cape(parcel_temperature, parcel_relative_humidity,
parcel_pressure)
    cape_values.append(cape)

# Cape values as a function of time
pressure_levels = range(1000, 399, -100)  # Set the pressure levels

# Plotting the Cape values as a function of time
lines = plt.plot(pressure_levels, cape_values)  # Provide pressure levels as
x-axis and cape_values as y-axis

legend_labels = []
```

```
for line, date in zip(lines, dates):
    color = line.get_color()
    legend_labels.append(f"\nDate: {np.datetime_as_string(date, unit='D')}")

plt.xlabel('Pressure (hPa)')  # X-axis label
plt.ylabel('CAPE')  # Y-axis label
plt.title('CAPE Profile')  # Title of the plot
plt.grid(True)  # Display grid lines

# Creating a legend with colors and dates displayed side by side
plt.legend(legend_labels, loc='upper center', bbox_to_anchor=(0.5, -0.15),
ncol=len(legend_labels))

# Saving the plot
plt.savefig('D:/output/cape/Calcualted_cape')

plt.show()


# In[91]:


# If the dew point temperature - surface temperature difference is positive,
it
#indicates that the dew point temperature is higher than the surface
#temperature, suggesting that fog formation is unlikely or challenging.
# If the dew point temperature - surface temperature difference is negative,
it
#indicates that the dew point temperature is lower than the surface
temperature,
#creating favorable conditions for fog formation.
""
c_fog = d2m_-t2m_

plt.figure(figsize=(10, 5))
plt.plot(c_fog['time'], c_fog.sel(latitude=37.76441, longitude=38.27629,
method='nearest'), label='Adiyaman')
plt.plot(c_fog['time'], c_fog.sel(latitude=37.158333, longitude=38.791668,
method='nearest'), label='Sanliurfa')

plt.xlabel('Time')
plt.ylabel('CFog')
plt.title('Time-Series of boundary_layer_conditions_of_fog in Adiyaman')
plt.grid(True)
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()  # Adjust layout to prevent cropping
plt.savefig('D:/output/boundary_layer_conditions_of_fog/boundary_layer_conditi
ons_of_fog')
```

```
plt.legend()

plt.show()


# In[123]:


dataset_temp = xr.open_dataset('D:/temp_cape.nc')
dataset_rh = xr.open_dataset('D:/rh_cape2.nc')


# In[124]:


file = 'D:/data.nc'
data = Dataset(file, mode='r')

print(type(data)) # xarray.core.dataarray.DataArray
print(data.variables.keys())  # get all variable names


# In[190]:


# If LI is a negative value, the atmosphere is potentially unstable and the
potential for
#thunderstorms increases. If it is a positive value, the atmosphere is more
stable and a
#thunderstorm is less likely to occur.

# Step 1: Load the temperature dataset
dataset_temp = xr.open_dataset('D:/temp_cape.nc')

# Step 2: Vertical profile of temperature and relative humidity for Adiyaman
temperature_profile_adiyaman = dataset_temp['t'].sel(latitude=37.76441,
longitude=38.27629, method='nearest')
temperature_500hPa_adiyaman = temperature_profile_adiyaman.sel(level=500)
temperature_difference_adiyaman = t2m_adiyaman - temperature_500hPa_adiyaman
times_adiyaman = np.array(temperature_difference_adiyaman.time.values,
dtype='datetime64[ns]')

# Step 3: Vertical profile of temperature and relative humidity for Sanliurfa
temperature_profile_sanliurfa = dataset_temp['t'].sel(latitude=37.158333,
longitude=38.791668, method='nearest')
temperature_500hPa_sanliurfa = temperature_profile_sanliurfa.sel(level=500)
temperature_difference_sanliurfa = t2m_urfa - temperature_500hPa_sanliurfa
times_sanliurfa = np.array(temperature_difference_sanliurfa.time.values,
dtype='datetime64[ns]')
```

```
# Plotting
plt.plot(times_adiyaman, temperature_difference_adiyaman, label='Adiyaman')
plt.plot(times_sanliurfa, temperature_difference_sanliurfa, label='Sanliurfa')
plt.xlabel('Time')
plt.ylabel('Temperature Difference (K)')
plt.title('Lifted Index')
plt.grid(True)
plt.xticks(rotation=45)  # Rotate date labels by 45 degrees
plt.legend()
plt.savefig('D:/output/LiftedIndex_combined.png')
plt.show()


# In[131]:


import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Calculation of CIN for Adiyaman
CIN_adiyaman = df_adiyaman['Theta'] - temperature_500hPa_adiyaman

# Calculation of CIN for Sanliurfa
CIN_sanliurfa = df_urfa['Theta'] - temperature_500hPa_sanliurfa

dates = df_adiyaman['Datetime']

fig, ax = plt.subplots()

# Plotting the data
ax.plot(dates, CIN_adiyaman, label='Adiyaman')
ax.plot(dates, CIN_sanliurfa, label='Sanliurfa')

date_fmt = mdates.DateFormatter('%m-%d')
ax.xaxis.set_major_formatter(date_fmt)

ax.set_xlabel('Date (Month-Day)')
ax.set_ylabel('CIN')
ax.set_title('CIN by Time (Hourly)')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.savefig('D:/output/CIN_combined.png')
plt.show()


# In[135]:
```

```
data_k = xr.open_dataset('D:/k_trh.nc')

# Adiyaman
time_dim = len(data_k['time'])
level_dim = len(data_k['level'])

latitude_adiyaman = 37.76441
longitude_adiyaman = 38.27629

lat_idx_adiyaman = np.abs(data_k['latitude'] - latitude_adiyaman).argmin()
lon_idx_adiyaman = np.abs(data_k['longitude'] - longitude_adiyaman).argmin()

k_index_adiyaman = np.zeros(time_dim)

for t in range(time_dim):
    t850 = data_k['t'][t, level_dim-1, lat_idx_adiyaman, lon_idx_adiyaman]
    r850 = data_k['r'][t, level_dim-1, lat_idx_adiyaman, lon_idx_adiyaman]
    t500 = data_k['t'][t, 0, lat_idx_adiyaman, lon_idx_adiyaman]
    r500 = data_k['r'][t, 0, lat_idx_adiyaman, lon_idx_adiyaman]

    k_index_adiyaman[t] = (t850 - t500) + r850 - r500

# Sanliurfa
latitude_sanliurfa = 37.158333
longitude_sanliurfa = 38.791668

lat_idx_sanliurfa = np.abs(data_k['latitude'] - latitude_sanliurfa).argmin()
lon_idx_sanliurfa = np.abs(data_k['longitude'] - longitude_sanliurfa).argmin()

k_index_sanliurfa = np.zeros(time_dim)

for t in range(time_dim):
    t850 = data_k['t'][t, level_dim-1, lat_idx_sanliurfa, lon_idx_sanliurfa]
    r850 = data_k['r'][t, level_dim-1, lat_idx_sanliurfa, lon_idx_sanliurfa]
    t500 = data_k['t'][t, 0, lat_idx_sanliurfa, lon_idx_sanliurfa]
    r500 = data_k['r'][t, 0, lat_idx_sanliurfa, lon_idx_sanliurfa]

    k_index_sanliurfa[t] = (t850 - t500) + r850 - r500

# Plotting
dates = data_k['time']

plt.plot(dates, k_index_adiyaman, label='Adiyaman')
plt.plot(dates, k_index_sanliurfa, label='Sanliurfa')
plt.xlabel('Time')
plt.ylabel('K Index')
plt.title('K Index by Time Hourly')
plt.grid(True)
plt.xticks(rotation=45)
```

```
plt.legend()
plt.savefig('D:/output/KIndex_combined.png')
plt.show()


# In[138]:


import numpy as np
import xarray as xr
import matplotlib.pyplot as plt

# Veri setini açın
data_index = xr.open_dataset('D:/index.nc')

# Adiyaman için istenilen konumun latitude ve longitude değerleri
latitude_adiyaman = 37.76441
longitude_adiyaman = 38.27629

# Sanliurfa için istenilen konumun latitude ve longitude değerleri
latitude_sanliurfa = 37.158333
longitude_sanliurfa = 38.791668

# İstenilen konum için en yakın lat/lon indekslerini bulun
lat_idx_adiyaman = np.abs(data_index['latitude'] - latitude_adiyaman).argmin()
lon_idx_adiyaman = np.abs(data_index['longitude'] -
longitude_adiyaman).argmin()

lat_idx_sanliurfa = np.abs(data_index['latitude'] -
latitude_sanliurfa).argmin()
lon_idx_sanliurfa = np.abs(data_index['longitude'] -
longitude_sanliurfa).argmin()

# İstenilen konumlar için T850 ve T500 değerlerini alın
t = data_index['t']
T850_adiyaman = t[:, 2, lat_idx_adiyaman, lon_idx_adiyaman]  # 850 hPa
seviyesi indeksi = 2
T500_adiyaman = t[:, 1, lat_idx_adiyaman, lon_idx_adiyaman]  # 500 hPa
seviyesi indeksi = 1

T850_sanliurfa = t[:, 2, lat_idx_sanliurfa, lon_idx_sanliurfa]  # 850 hPa
seviyesi indeksi = 2
T500_sanliurfa = t[:, 1, lat_idx_sanliurfa, lon_idx_sanliurfa]  # 500 hPa
seviyesi indeksi = 1

# İstenilen konumlar için TD850 ve TD500 değerlerini alın
td = data_index['r']  # Çiğ noktası verisi
TD850_adiyaman = td[:, 2, lat_idx_adiyaman, lon_idx_adiyaman]  # 850 hPa
seviyesi indeksi = 2
```

```
TD500_adiyaman = td[:, 1, lat_idx_adiyaman, lon_idx_adiyaman]  # 500 hPa
seviyesi indeksi = 1

TD850_sanliurfa = td[:, 2, lat_idx_sanliurfa, lon_idx_sanliurfa]  # 850 hPa
seviyesi indeksi = 2
TD500_sanliurfa = td[:, 1, lat_idx_sanliurfa, lon_idx_sanliurfa]  # 500 hPa
seviyesi indeksi = 1

# İstenilen konumlar için U (u bileşeni) ve V (v bileşeni) vektörlerini alın
u_vector = data_index['u']
v_vector = data_index['v']

# Adiyaman için 850 hPa seviyesindeki rüzgar hızını (U850) hesaplayın
U850_adiyaman = np.sqrt(u_vector[:, 2, lat_idx_adiyaman, lon_idx_adiyaman]**2
+
                        v_vector[:, 2, lat_idx_adiyaman,
lon_idx_adiyaman]**2)  # 850 hPa seviyesi indeksi = 2

# Adiyaman için 500 hPa seviyesindeki rüzgar hızını (U500) hesaplayın
U500_adiyaman = np.sqrt(u_vector[:, 1, lat_idx_adiyaman, lon_idx_adiyaman]**2
+
                        v_vector[:, 1, lat_idx_adiyaman,
lon_idx_adiyaman]**2)  # 500 hPa seviyesi indeksi = 1

# Sanliurfa için 850 hPa seviyesindeki rüzgar hızını (U850) hesaplayın
U850_sanliurfa = np.sqrt(u_vector[:, 2, lat_idx_sanliurfa,
lon_idx_sanliurfa]**2 +
                         v_vector[:, 2, lat_idx_sanliurfa,
lon_idx_sanliurfa]**2)  # 850 hPa seviyesi indeksi = 2

# Sanliurfa için 500 hPa seviyesindeki rüzgar hızını (U500) hesaplayın
U500_sanliurfa = np.sqrt(u_vector[:, 1, lat_idx_sanliurfa,
lon_idx_sanliurfa]**2 +
                         v_vector[:, 1, lat_idx_sanliurfa,
lon_idx_sanliurfa]**2)  # 500 hPa seviyesi indeksi = 1

# Adiyaman için TTI hesaplama
delta_T_adiyaman = T850_adiyaman - T500_adiyaman
delta_TD_adiyaman = TD850_adiyaman - TD500_adiyaman

delta_U_adiyaman = U850_adiyaman - U500_adiyaman
TTI_adiyaman = delta_T_adiyaman + delta_TD_adiyaman + delta_U_adiyaman
# Sanliurfa için TTI hesaplama
delta_T_sanliurfa = T850_sanliurfa - T500_sanliurfa
delta_TD_sanliurfa = TD850_sanliurfa - TD500_sanliurfa
delta_U_sanliurfa = U850_sanliurfa - U500_sanliurfa
TTI_sanliurfa = delta_T_sanliurfa + delta_TD_sanliurfa + delta_U_sanliurfa

dates = data_index['time']
```

```
# TTI değerlerini alın
TTI_values_adiyaman = delta_T_adiyaman + delta_TD_adiyaman + delta_U_adiyaman
TTI_values_sanliurfa = delta_T_sanliurfa + delta_TD_sanliurfa +
delta_U_sanliurfa

# Grafik çizimi
plt.plot(dates, TTI_values_adiyaman, label='Adiyaman')
plt.plot(dates, TTI_values_sanliurfa, label='Sanliurfa')
plt.xlabel('Time')
plt.ylabel('TTI')
plt.title('TTI by Time ( Hourly)')
plt.grid(True)
plt.xticks(rotation=45)  # Tarih etiketlerini 45 derece döndürme
plt.legend()
plt.savefig('D:/output/TTI_combined.png')
plt.show()


# In[139]:


# temperature data loop for 7 days
times = np.arange(0,168)                # 7 days, 24 hours each

# creating meshgrid from our list of spatial coordinates
lon, lat = np.meshgrid(longs,lats)          # create a mesh of our coordinates

# generate a frame for each day

for i in times:
    # set the axes' spatial projection, styling, and title
    ax = plt.axes(projection=ccrs.PlateCarree())            # set the
projection
    ax.coastlines()                     # add coastlines

    if i < 24:
        ax.set_title('Temperature (C) on date 2023-03-11 (Hourly)')  # set the
title
    elif i < 48:
        ax.set_title('Temperature (C) on date 2023-03-12 (Hourly)')
    elif i < 72:
        ax.set_title('Temperature (C) on date 2023-03-13 (Hourly)')
    elif i < 96:
        ax.set_title('Temperature (C) on date 2023-03-14 (Hourly)')
    elif i < 120:
        ax.set_title('Temperature (C) on date 2023-03-15 (Hourly)')
    elif i < 143:
        ax.set_title('Temperature (C) on date 2023-03-16 (Hourly)')
```

```
    else:
        ax.set_title('Temperature (C) on date 2023-03-17 (Hourly)')

    # generate the contour map

    plt.contourf(longs, lats, t2m[i,],60, transform=ccrs.PlateCarree(),
cmap='hot') # contour map
    plt.colorbar()                    # add a colorbar


    # save the figure and give a pause for visualization
    plt.savefig(f'D:/output/t2m/{i+1}.png')
    plt.pause(1)
plt.show()


# In[145]:


image_frames = []   # empty list to store frames
days = np.arange(1,168) # sub-sample time number 1 to 168

for k in days:
    new_frame = PIL.Image.open(f'D:/output/t2m/{k}.png')    # open each image
    image_frames.append(new_frame)                         # append to the list

# save as GIF
image_frames[0].save('D:/output/t2m/t2m.gif', format='GIF',     # save as GIF
                     append_images=image_frames[1:],      # append frames
                     save_all=True,                      # save all frames
                     duration=150, loop=0)             # frame duration 300ms


# In[146]:


#GIF Speed Up
gif_original = 'D:/output/t2m/t2m.gif'  # original GIF
gif_speed_up = 'D:/output/t2m/t2m_speedup.gif'      # speed up GIF

gif = imageio.mimread(gif_original)     # read the original GIF

imageio.mimsave(gif_speed_up, gif, fps=15)  # save the speed up GIF


# In[147]:


# loop for 7 days
times = np.arange(0,168)
```

```
# creating meshgrid from our list of spatial coordinates
lon, lat = np.meshgrid(longs,lats)

# generate a frame for each day

for i in times:
    # set the axes' spatial projection, styling, and title
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.coastlines()

    if i < 25:
        ax.set_title('Relative Humidity (%) on date 2023-03-11 (Hourly)')
    elif i < 48:
        ax.set_title('Relative Humidity (%)) on date 2023-03-12 (Hourly)')
    elif i < 72:
        ax.set_title('Relative Humidity (%) on date 2023-03-13 (Hourly)')
    elif i < 96:
        ax.set_title('Relative Humidity (%) on date 2023-03-14 (Hourly)')
    elif i < 120:
        ax.set_title('Relative Humidity (%) on date 2023-03-15 (Hourly)')
    elif i < 144:
        ax.set_title('Relative Humidity (%) on date 2023-03-16 (Hourly)')
    else:
        ax.set_title('Relative Humidity (%) on date 2023-03-17 (Hourly)')

    # generate the contour map

    plt.contourf(longs, lats, RH[i,],60, transform=ccrs.PlateCarree(),
cmap='GnBu')
    plt.colorbar()

    # save the figure and give a pause for visualization
    plt.savefig(f'D:/output/RH/{i+1}.png')
    plt.pause(1)
plt.show()


# In[148]:


image_frames = []
days = np.arange(1,168) # sub-sample time number 1 to 168

for k in days:
    new_frame = PIL.Image.open(f'D:/output/RH/{k}.png')
    image_frames.append(new_frame)

# save as GIF
```

```
image_frames[0].save('D:/output/RH/RH.gif', format='GIF',
                     append_images=image_frames[1:],
                     save_all=True,
                     duration=150, loop=0)


# In[149]:


#GIF Speed Up
gif_original = 'D:/output/RH/RH.gif'
gif_speed_up = 'D:/output/RH/RH_speedup.gif'

gif = imageio.mimread(gif_original)

imageio.mimsave(gif_speed_up, gif, fps=15)


# In[151]:


# dew-point temperature data loop for 7 days
times = np.arange(0,168)                # 7 days, 24 hours each

# creating meshgrid from our list of spatial coordinates
lon, lat = np.meshgrid(longs,lats)          # create a mesh of our coordinates

# generate a frame for each day

for i in times:
    # set the axes' spatial projection, styling, and title
    ax = plt.axes(projection=ccrs.PlateCarree())            # set the
projection
    ax.coastlines()                     # add coastlines

    if i < 25:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-11
(Hourly)')  # set the title
    elif i < 48:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-12 (Hourly)')
    elif i < 72:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-13 (Hourly)')
    elif i < 96:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-14 (Hourly)')
    elif i < 120:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-15 (Hourly)')
    elif i < 144:
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-16 (Hourly)')
    else:
```

```
        ax.set_title('Dew-Point Temperature (C) on date 2023-03-17 (Hourly)')

    # generate the contour map

    plt.contourf(longs, lats, d2m[i,], transform=ccrs.PlateCarree(),
cmap='viridis') # contour map

    plt.colorbar()                    # add a colorbar

    # save the figure and give a pause for visualization
    plt.savefig(f'D:/output/d2m/{i+1}.png')
    plt.pause(1)
plt.show()


# In[152]:


image_frames = []
days = np.arange(1,168) # sub-sample time number 1 to 168

for k in days:
    new_frame = PIL.Image.open(f'D:/output/d2m/{k}.png')
    image_frames.append(new_frame)

# save as GIF
image_frames[0].save('D:/output/d2m/d2m.gif', format='GIF',
                     append_images=image_frames[1:],
                     save_all=True,
                     duration=150, loop=0)


# In[153]:


#GIF Speed Up
gif_original = 'D:/output/d2m/d2m.gif'
gif_speed_up = 'D:/output/d2m/d2m_speedup.gif'

gif = imageio.mimread(gif_original)

imageio.mimsave(gif_speed_up, gif, fps=15)


# In[6]:


import matplotlib.colors as mcolors

# 10m wind data loop for 7 days
```

```
times = np.arange(0, 168)  # 7 days, 24 hours each

lon, lat = np.meshgrid(longs, lats)  # create a mesh of our coordinates

# Calculate the wind speed magnitude
wind_speed = np.sqrt(u10**2 + v10**2)

# Normalize wind speed for color mapping
norm = mcolors.Normalize(vmin=wind_speed.min(), vmax=wind_speed.max())

for i in times:
    fig = plt.figure()  # create a figure
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())  # set the
projection

    # Plot the wind vectors with color mapping
    quiver = ax.quiver(lon, lat, u10[i, :, :], v10[i, :, :], wind_speed[i, :,
:], transform=ccrs.PlateCarree(), cmap='twilight', norm=norm)

    ax.coastlines()  # add coastlines
    ax.gridlines(draw_labels=True)  # add gridlines with labels

    plt.colorbar(quiver)  # add a colorbar based on wind speed

    if i < 24:
        plt.title('Wind Speed on date 2023-03-11')  # set the title
    elif i < 48:
        plt.title('Wind Speed on date 2023-03-12')
    elif i < 72:
        plt.title('Wind Speed on date 2023-03-13')
    elif i < 96:
        plt.title('Wind Speed on date 2023-03-14')
    elif i < 120:
        plt.title('Wind Speed on date 2023-03-15')
    elif i < 143:
        plt.title('Wind Speed on date 2023-03-16')
    else:
        plt.title('Wind Speed on date 2023-03-17')

    plt.savefig(f'D:/output/wind/{i+1}.png')  # save the figure
    plt.pause(1)  # give a pause for visualization

plt.show()  # show the figure


# In[31]:


import matplotlib.colors as mcolors
```

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# 10m wind data loop for 7 days
times = np.arange(0, 168)  # 7 days, 24 hours each

lon, lat = np.meshgrid(longs, lats)  # create a mesh of our coordinates

# Calculate the wind speed magnitude
wind_speed = np.sqrt(u10**2 + v10**2)

# Normalize wind speed for color mapping
norm = mcolors.Normalize(vmin=wind_speed.min(), vmax=wind_speed.max())

for i in times:
    fig = plt.figure(figsize=(12, 6))  # create a figure with desired size

    ax1 = fig.add_subplot(1, 2, 1, projection=ccrs.PlateCarree())  # set the
projection for the first subplot

    # Plot the wind vectors with color mapping
    quiver = ax1.quiver(lon, lat, u10[i, :, :], v10[i, :, :], wind_speed[i, :,
:],
                        transform=ccrs.PlateCarree(), cmap='twilight', norm=norm,
                        width=0.01, scale=100)

    ax1.coastlines()  # add coastlines
    ax1.gridlines(draw_labels=False)  # remove gridlines with labels

    # Specify the latitude and longitude range for the zoomed-in map
    lat_min = 35
    lat_max = 41
    lon_min = 35
    lon_max = 43

    # Set the extent of the zoomed-in map
    ax1.set_extent([lon_min, lon_max, lat_min, lat_max])

    # Add additional map features
    ax1.add_feature(cfeature.LAND, facecolor='lightgray')
    ax1.add_feature(cfeature.OCEAN)
    ax1.add_feature(cfeature.COASTLINE)
    ax1.add_feature(cfeature.BORDERS, linestyle=':')

    if i < 24:
        plt.title('Wind Speed on date 2023-03-11 (Hourly)')  # set the title
    elif i < 48:
        plt.title('Wind Speed on date 2023-03-12 (Hourly)')
    elif i < 72:
```

```
        plt.title('Wind Speed on date 2023-03-13 (Hourly)')
    elif i < 96:
        plt.title('Wind Speed on date 2023-03-14 (Hourly)')
    elif i < 120:
        plt.title('Wind Speed on date 2023-03-15 (Hourly)')
    elif i < 143:
        plt.title('Wind Speed on date 2023-03-16 (Hourly)')
    else:
        plt.title('Wind Speed on date 2023-03-17 (Hourly)')

    ax2 = fig.add_subplot(1, 2, 2, projection=ccrs.PlateCarree())  # set the
projection for the second subplot

    # Plot the wind vectors with color mapping
    quiver = ax2.quiver(lon, lat, u10[i, :, :], v10[i, :, :], wind_speed[i, :,
:],
                        transform=ccrs.PlateCarree(), cmap='twilight', norm=norm,
                        width=0.01, scale=100)

    ax2.coastlines()  # add coastlines
    ax2.gridlines(draw_labels=False)  # remove gridlines with labels

    if i < 24:
        plt.title('Wind Speed on date 2023-03-11 (Hourly)')  # set the title
    elif i < 48:
        plt.title('Wind Speed on date 2023-03-12 (Hourly)')
    elif i < 72:
        plt.title('Wind Speed on date 2023-03-13 (Hourly)')
    elif i < 96:
        plt.title('Wind Speed on date 2023-03-14 (Hourly)')
    elif i < 120:
        plt.title('Wind Speed on date 2023-03-15 (Hourly)')
    elif i < 143:
        plt.title('Wind Speed on date 2023-03-16 (Hourly)')
    else:
        plt.title('Wind Speed on date 2023-03-17 (Hourly)')

    # Create colorbar axes and adjust their position
    cax = fig.add_axes([0.125, 0.1, 0.775, 0.03])  # updated position of
colorbar
    cbar = plt.colorbar(quiver, cax=cax, orientation='horizontal')  # add a
horizontal colorbar based on wind speed

    # Increase the number of colorbar ticks and spacing
    num_ticks = 8  # number of ticks on colorbar
    spacing = (wind_speed.max() - wind_speed.min()) / num_ticks  # spacing
between ticks
    ticks = np.arange(wind_speed.min(), wind_speed.max() + spacing,
spacing)  # generate ticks
```

```
    cbar.set_ticks(ticks)  # set the ticks on colorbar

    plt.savefig(f'D:/output/wind/{i+1}.png')  # save the figure
    plt.close(fig)  # close the figure to free memory

plt.show()  # show the figures


# In[32]:


image_frames = []
days = np.arange(1,168) # sub-sample time number 1 to 168

for k in days:
    new_frame = PIL.Image.open(f'D:/output/wind/{k}.png')
    image_frames.append(new_frame)

# save as GIF
image_frames[0].save('D:/output/wind/wind.gif', format='GIF',
                     append_images=image_frames[1:],
                     save_all=True,
                     duration=200, loop=0)


# In[33]:


#GIF Speed Up
image_frames = []
days = np.arange(1,168) # sub-sample time number 1 to 168

for k in days:
    new_frame = PIL.Image.open(f'D:/output/wind/{k}.png')
    image_frames.append(new_frame)

# save as GIF
image_frames[0].save('D:/output/wind/wind_speedup.gif', format='GIF',
                     append_images=image_frames[1:],
                     save_all=True,
                     duration=120, loop=0)
```