Applications of Machine Learning in Remote Sensing

Homework 4

Nakul Poudel – np1140@rit.edu https://github.com/np1140/MLRemoteSensing

```
!pip install earthengine-api # Installing Google Earth Engine
```

```
Requirement already satisfied: earthengine-api in /usr/local/lib/python3.12/dist-packages (1.5.24)
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.12/dist-packages (from earthengine-
Requirement already satisfied: google-api-python-client>=1.12.1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: google-auth>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from earthengine-ap
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.12/dist-packages (from earth
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.12/dist-packages (from earthengine
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from earthengine-api) (2.32.4
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0,>=1.31.5 in /usr/local/lib/p
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.12/dist-packages (from google-api-
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from google-aut
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth>=1.4.1-
Requirement already satisfied: pyparsing<4,>=3.0.4 in /usr/local/lib/python3.12/dist-packages (from httplib2<1dev
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: google-resumable-media>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from goo
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.12/dist-packages (from google
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->earthengin
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->eart
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->eart
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<7
Requirement already satisfied: proto-plus<2.0.0,>=1.22.3 in /usr/local/lib/python3.12/dist-packages (from google-
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modul
```

```
!earthengine authenticate  # Authenticating to Earth Engine
```

```
essages before absl::InitializeLog() is called are written to STDERR
0478.517957    1416 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register factory for plugin
0478.541514    1416 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to register factory for plug
0478.595668    1416 computation_placer.cc:177] computation placer already registered. Please check linkage and av
0478.595710    1416 computation_placer.cc:177] computation placer already registered. Please check linkage and av
0478.595715    1416 computation_placer.cc:177] computation placer already registered. Please check linkage and av
0478.595719    1416 computation_placer.cc:177] computation placer already registered. Please check linkage and av
ted support in Colab. Use ee.Authenticate() or --auth_mode=notebook instead.
773 140233752166400 _default.py:711] No project ID could be determined. Consider running `gcloud config set proje
s needed by Earth Engine, open the following URL in a web browser and follow the instructions. If the web browser

arthengine.google.com/client-auth?scopes=https%3A//www.googleapis.com/auth/earthengine%20https%3A//www.googleapis

workflow will generate a code, which you should paste in the box below.
 code: 4/1Ab32j92jdKDR1I0SHwbdDCsjb-epQmDL8-ZehM2rQS9jHOMt27DQ99P3I7s

 authorization token.
```

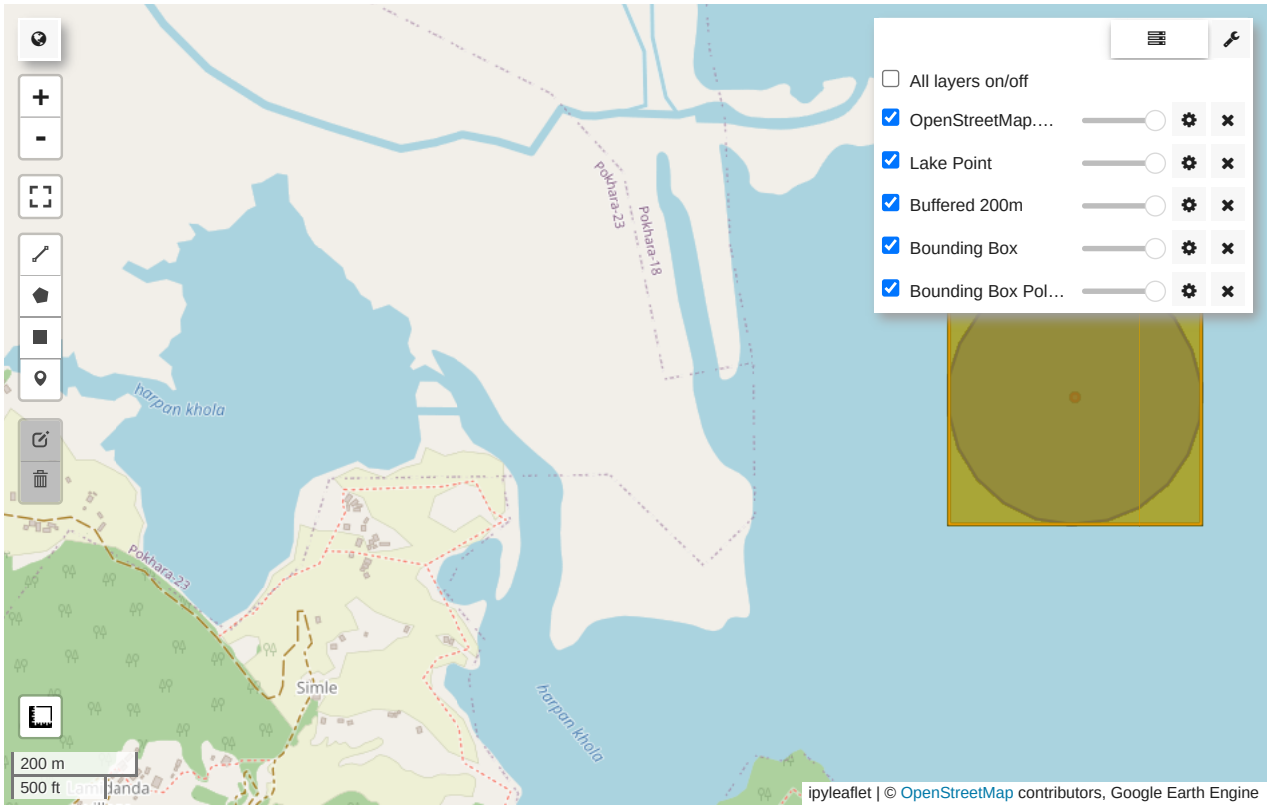
```
import ee
ee.Initialize()   # Importing and Initializing Earth Engine
```

```
# !pip install geemap  # Visualizing an Image Using Geemap
# import geemap
```

```
#Lake:  Lake Fewa (approx. center coordinates)
# Latitude: 28.211, Longitude: 83.9497
lake_point = ee.Geometry.Point([83.94, 28.22])

# Create a Region of Interest (ROI)
# Buffer by 200 meters
lake_buffer = lake_point.buffer(200)

# Get the bounding box of this buffer
lake_bounds = lake_buffer.bounds()
lake_polygon = ee.Geometry.Polygon(lake_bounds.coordinates())

# Print the geometries to the console
print("Lake Point:", lake_point.getInfo())
print("Buffered Area (200m):", lake_buffer.getInfo())
print("Bounding Box:", lake_bounds.getInfo())

# Display in the Earth Engine Code Editor Map
```

```
Map = geemap.Map()
Map.centerObject(lake_point, 15)
Map.addLayer(lake_point, {'color': 'red'}, 'Lake Point')
Map.addLayer(lake_buffer, {'color': 'blue'}, 'Buffered 200m')
Map.addLayer(lake_bounds, {'color': 'green'}, 'Bounding Box')
Map.addLayer(lake_polygon, {'color': 'orange'}, 'Bounding Box Polygon')

Map
```

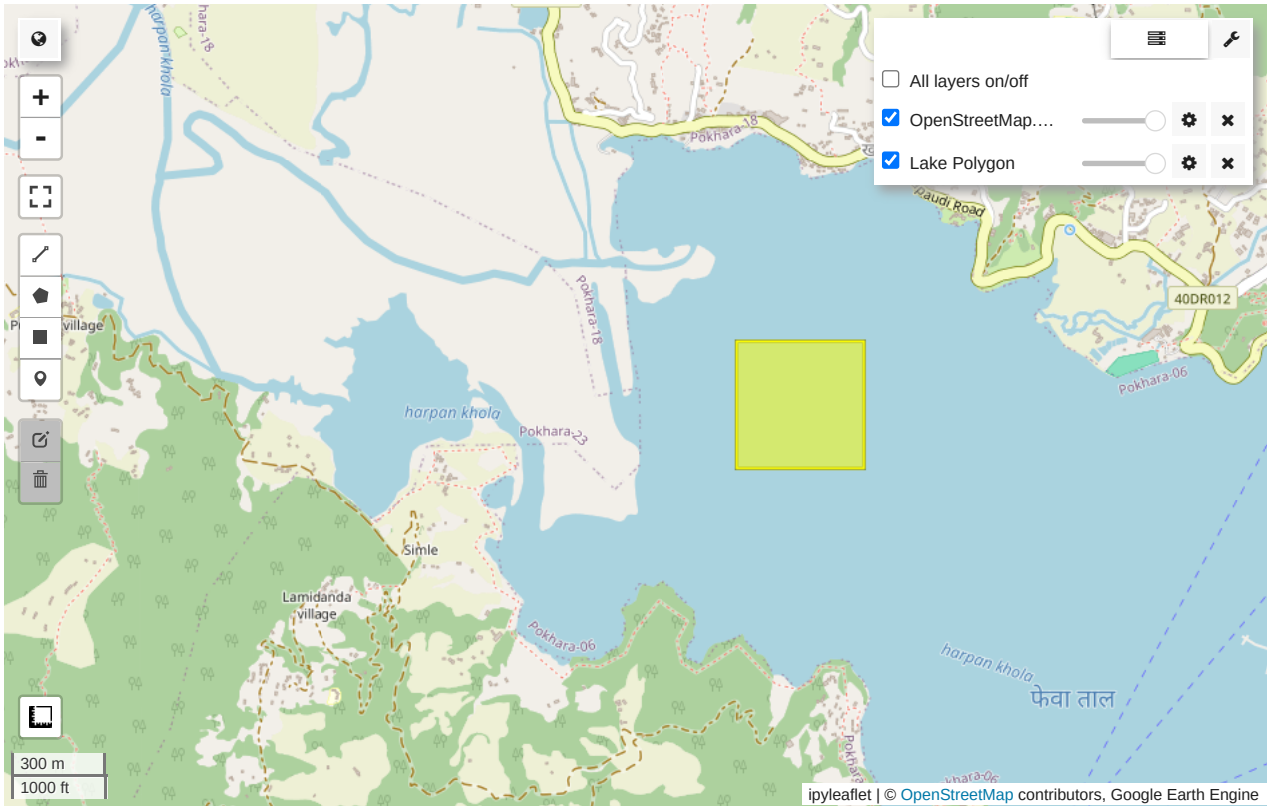
```
Lake Point: {'type': 'Point', 'coordinates': [83.94, 28.22]}
Buffered Area (200m): {'type': 'Polygon', 'coordinates': [[[83.94, 28.221799773190305], [83.93942947801438, 28.22
Bounding Box: {'geodesic': False, 'type': 'Polygon', 'coordinates': [[[83.93797407572619, 28.218200918436708], [8
```




```
import pandas as pd
landsat_name = 'LANDSAT/LC09/C02/T1_L2'
landsat8_collection = ee.ImageCollection(landsat_name) \
    .filterBounds(lake_polygon) \
    .filterDate('2021-10-31', '2025-10-17')  # date range

# Surface Temperature scale & offset
ST_SCALE = 0.00341802
ST_OFFSET = 149.0

# Function to extract ST band and convert to Kelvin

def add_LST_band(image):
    bands = image.bandNames()
    st_bands = bands.filter(ee.Filter.stringContains('item', 'ST_'))
    has_st = st_bands.size().gt(0)

    def scale_ST():
        st_img = image.select([st_bands.get(0)]).multiply(ST_SCALE).add(ST_OFFSET).rename('LST_K')
        return image.addBands(st_img).set('hasST', True)

    def no_ST():
        return image.set('hasST', False)

    return ee.Image(ee.Algorithms.If(has_st, scale_ST(), no_ST())).copyProperties(image)

# Map function over collection

landsat8_lst = landsat8_collection.map(add_LST_band).filter(ee.Filter.eq('hasST', True)) \
    .sort('system:time_start')

print("Total Landsat 8 scenes with ST:", landsat8_lst.size().getInfo())

# visualize an example scene using geemap
m = geemap.Map(center=[28.22, 83.94], zoom=15)
```

```
first_image = ee.Image(landsat8_lst.first())
m.addLayer(lake_polygon, {'color': 'yellow'}, 'Lake Polygon')
m
```

```
Total Landsat 8 scenes with ST: 166
```




```
def mask_clouds_and_snow(image):
    # Select the QA_PIXEL band
    qa = image.select('QA_PIXEL')

    # Bits to check
    cloud_shadow_bit = 1 << 3
    snow_bit = 1 << 4
    cloud_bit = 1 << 5
    cirrus_bit = 1 << 9

    # Create a mask for all bad pixels
    mask = qa.bitwiseAnd(cloud_shadow_bit)\
             .Or(qa.bitwiseAnd(snow_bit))\
             .Or(qa.bitwiseAnd(cloud_bit))\
             .Or(qa.bitwiseAnd(cirrus_bit))\
             .eq(0)  # Keep only clear pixels

    # Apply the mask to all bands
    return image.updateMask(mask)
```

```
landsat8_lst_masked = landsat8_lst.map(mask_clouds_and_snow)
print("Total Landsat 8 scenes with ST and no clouds:", landsat8_lst_masked.size().getInfo())
```

```
Total Landsat 8 scenes with ST and no clouds: 86
```

```
def compute_mean_lst(image):
    # Select the correct LST band
    lst = image.select('LST_K')

    # Compute mean over ROI
    mean_dict = lst.reduceRegion(
        reducer=ee.Reducer.mean(),
        geometry=lake_polygon,
        scale=30,
        maxPixels=1e9
    )

    mean_value = ee.Number(mean_dict.get('LST_K'))
```

```
    # Return as Feature
    return ee.Feature(None, {
        'date': image.date().format('YYYY-MM-dd'),
        'mean_LST_K': mean_value
    })
```

```
lst_features = ee.FeatureCollection(
    landsat8_lst_masked.map(compute_mean_lst)
).filter(ee.Filter.notNull(['mean_LST_K']))
```

```
lst_features
```

```
▶ FeatureCollection LANDSAT/LC09/C02/T1_L2 (21 elements, 0 columns)
```

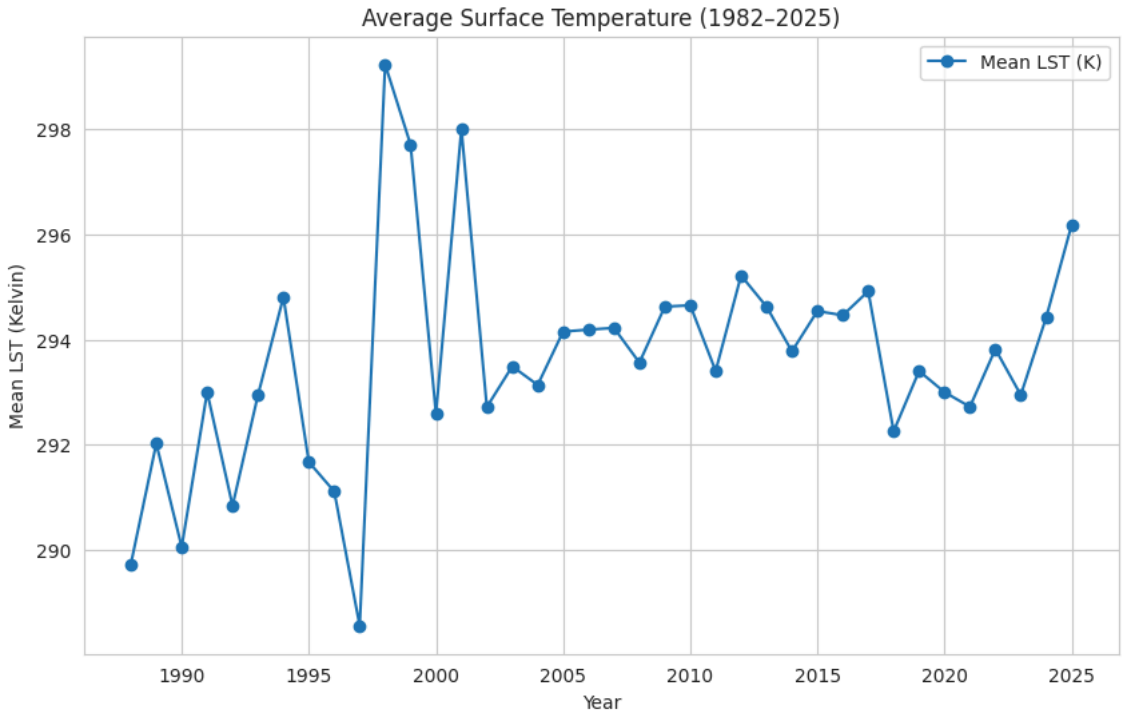
```
df = geemap.ee_to_df(lst_features) # convert to Pandas
df["landsat"] = landsat_name

print(df.head())
```

```
         date  mean_LST_K                landsat
0  2022-05-18  276.436197  LANDSAT/LC09/C02/T1_L2
1  2022-06-19  284.381019  LANDSAT/LC09/C02/T1_L2
2  2022-07-05  286.116726  LANDSAT/LC09/C02/T1_L2
3  2022-10-25  275.863805  LANDSAT/LC09/C02/T1_L2
4  2022-11-26  272.585918  LANDSAT/LC09/C02/T1 L2
```

```
import os

# CSV file name
csv_file = "/content/drive/MyDrive/ML remotesensing/HW4/tilicho_data.csv"

# Check if file exists
if not os.path.isfile(csv_file):
    df.to_csv(csv_file, index=False)
else:
    df.to_csv(csv_file, mode='a', index=False, header=False)
```

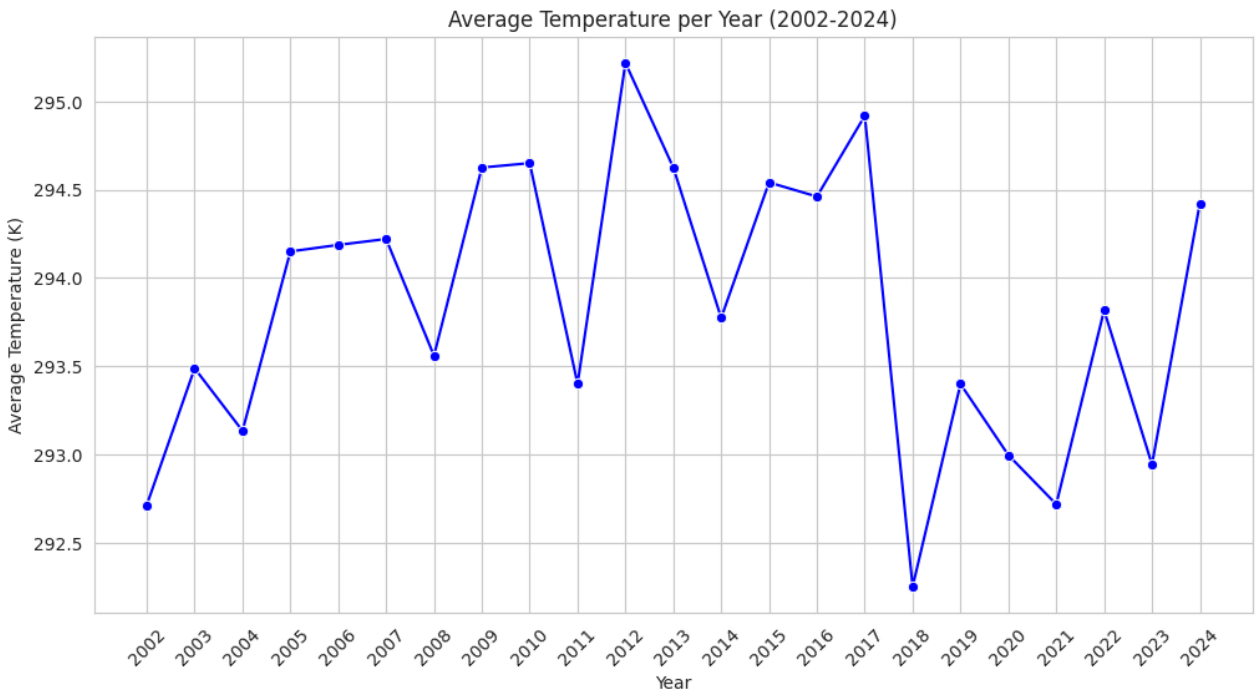
Visualization

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

# Load CSV
df = pd.read_csv("/content/drive/MyDrive/ML remotesensing/HW4/fewa_lst_data.csv")

# Convert date to datetime and extract year
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year

# Compute mean LST per year
yearly_mean = df.groupby('year')['mean_LST_K'].mean().reset_index()

# Plot
plt.figure(figsize=(10,6))
plt.plot(yearly_mean['year'], yearly_mean['mean_LST_K'], marker='o', linestyle='-', label='Mean LST (K)')
plt.title("Average Surface Temperature (1982–2025)")
plt.xlabel("Year")
plt.ylabel("Mean LST (Kelvin)")
plt.grid(True)
plt.legend()
plt.show()
```



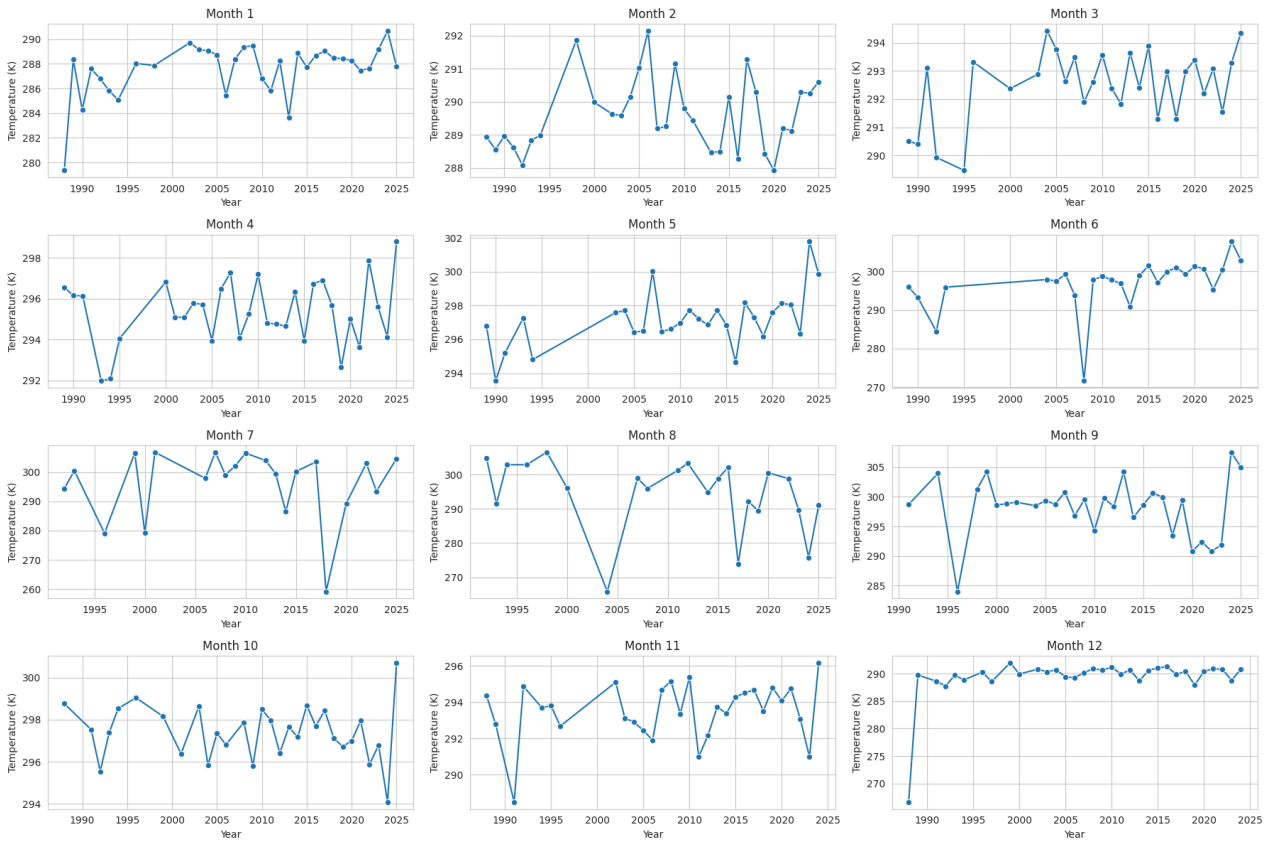
Temperature Trends over Well-Represented Years

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

# Load CSV
csv_file = "/content/drive/MyDrive/ML remotesensing/HW4/fewa_lst_data.csv"
df = pd.read_csv(csv_file)

df['date'] = pd.to_datetime(df['date'])

# Extract year
df['year'] = df['date'].dt.year

# Filter for years 2002 to 2024
df_filtered = df[(df['year'] >= 2002) & (df['year'] <= 2024)]

# Compute average temperature per year
avg_temp_per_year = df_filtered.groupby('year')['mean_LST_K'].mean().reset_index()

# Plot
sns.set_style("whitegrid")
plt.figure(figsize=(12, 6))
sns.lineplot(x='year', y='mean_LST_K', data=avg_temp_per_year, marker='o', color='b')
plt.title('Average Temperature per Year (2002-2024)')
plt.xlabel('Year')
plt.ylabel('Average Temperature (K)')
plt.xticks(avg_temp_per_year['year'], rotation=45)
plt.show()
```



The plot shows the average yearly temperature over time of Lake Fewa in Nepal. Data from Landsat 4 and 5 are sparse, making the early averages less reliable. Furthermore, since temperature varies throughout the year and only a few months are recorded in these early datasets, the averages may not fully reflect annual conditions. In contrast, data from Landsat 7, 8, and 9 after 2000 provide more comprehensive coverage, making the yearly averages more representative. Observing the period from 2002 onward, there is a slight increase in the lake's temperature, with a cooler period between 2018 and 2021. From 2022 onward, temperatures rise again, reaching above 294 K in 2023. While this suggests a possible warming trend, it is not sufficient to conclusively confirm global warming for this lake.

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

# Load CSV
csv_file = "/content/drive/MyDrive/ML remotesensing/HW4/fewa_lst_data.csv"
df = pd.read_csv(csv_file)

df['date'] = pd.to_datetime(df['date'])

# Extract year and month
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month

# Handle missing values
monthly_avg = df.groupby(['year', 'month'])['mean_LST_K'].mean().reset_index()

# Plotting style
sns.set_style("whitegrid")

# 12 plots, one per month
fig, axes = plt.subplots(4, 3, figsize=(18, 12))
axes = axes.flatten()

for month in range(1, 13):
    ax = axes[month-1]
    data = monthly_avg[monthly_avg['month'] == month]
    sns.lineplot(x='year', y='mean_LST_K', data=data, ax=ax, marker='o')
    ax.set_title(f'Month {month}')
    ax.set_xlabel('Year')
    ax.set_ylabel('Temperature (K)')

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



In the plot above, we also visualized the monthly temperature variation from 1988 to the present.

```
import pandas as pd

# Load CSV
csv_file = "/content/drive/MyDrive/ML remotesensing/HW4/fewa_lst_data.csv"
df = pd.read_csv(csv_file)

df['date'] = pd.to_datetime(df['date'])

# Extract year
df['year'] = df['date'].dt.year

# Count number of data points per year for all years in the dataset
data_count_per_year = df.groupby('year').size().reset_index(name='data_count')

print(data_count_per_year)
```

```
    year  data_count
0   1988          11
1   1989          21
2   1990          11
3   1991          26
4   1992          21
5   1993          26
6   1994          21
7   1995           8
8   1996          14
9   1997           1
10  1998           6
11  1999           9
12  2000          11
13  2001          12
14  2002          11
15  2003          22
16  2004          43
17  2005          39
```

```
import pandas as pd

# Load CSV
csv_file = "/content/drive/MyDrive/ML remotesensing/HW4/fewa_lst_data.csv"
df = pd.read_csv(csv_file)

df['date'] = pd.to_datetime(df['date'])

# Extract year and month
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month

# Count unique months per year
months_per_year = df.groupby('year')['month'].nunique().reset_index()
months_per_year.columns = ['year', 'unique_months']

# Filter years with all 12 months
complete_years = months_per_year[months_per_year['unique_months'] == 12]['year'].tolist()

print("Years with data for all 12 months:")
print(complete_years)
```