Histrorical Trend Analysis of Lake Temperatures using LANDSAT Data obtained through Google Earth Engine (GEE).

Part I: Lake Temperature Data Collection

Author: Ramesh Bhatta, CIS, RIT

This Notebook goes through the steps to obtain the **historical temperatures** of any **Lake** around the world based on multiple **LANDSATs** (4, 5, 7, 8, and 9 till present time) and save the data in **time-series** csv file. The only data we need is the **latitude** and **longitude** of a point within the lake.

Note: This notebook is prepared to run on Google Collab. Feel free to play with the code for running locally or any other platforms.

```
## Installing Google Earth Engine ---> Needed to access GEE.

!pip install earthengine-api
```

```
## Mount the Google Drive to save and load data

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
## Authenticate

"""

    - Authentication token is required to use GEE to gain access to cloud storage and other resources.
    - Run this code and open the generated link to generate a Auth. token.
    - This needs users to have signed up for [GEE](https://signup.earthengine.google.com/).
     So make sure to sign up before running the code.

"""

!earthengine authenticate
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1743451153.516157     666 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one
E0000 00:00:1743451153.523434     666 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when



Authenticate: Limited support in Colab. Use ee.Authenticate() or --auth_mode=notebook instead.
W0331 19:59:20.966219 132851240034304 _default.py:711] No project ID could be determined. Consider running `gcloud config set project` or setting
To authorize access needed by Earth Engine, open the following URL in a web browser and follow the instructions. If the web browser does not start

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

The authorization workflow will generate a code, which you should paste in the box below.
Enter verification code: 4/1AQSTgQFyQ2jvGvFVD62ihg-F10keeEdH56gxlFEG1N4JjaPi4OrL4bfkn_Q

Successfully saved authorization token.
```

```
## Now we can Import and Initialize GEE module

import ee
ee.Initialize()
```

Lake Location

For this task I will be using **Lake Fewa: (Lat: 28.217602, Lon: 83.945385)** present in **Pokhara** city in **Nepal**. More details about the lake is included in **Part II: Data Analysis** present in *data_analysis.ipynb* file. This file is used just to generate time-series lake temperature data.

```
## Create a point geometry using latitude and longitude of the lake.

lake_point = ee.Geometry.Point([83.945385, 28.217602])
lake_point.getInfo()
```

```
{'type': 'Point', 'coordinates': [83.945385, 28.217602]}
```

```
## Buffer the point feature by 300 meters --> This will create a circular ROI around the point in the lake.

buffer_poly = lake_point.buffer(300)

## Buffer info

buffer_poly.getInfo()
```

```
{'type': 'Polygon',
 'coordinates': [[[83.945385, 28.22030165978545],
   [83.9445292293173, 28.220194210663333],
   [83.94374158281087, 28.21988041686614],
   [83.94308476046517, 28.21938525804401],
   [83.94261104611704, 28.21874815102861],
   [83.94235814541967, 28.218019811570006],
```

```
        [83.94234618509972, 28.21725821672896],
        [83.94257611217098, 28.216523989516016],
        [83.94302962012104, 28.21587557328085],
        [83.9436706074569, 28.215364579976356],
        [83.94444805208884, 28.215031682464897],
        [83.94530007253202, 28.214903377646362],
        [83.94615885270274, 28.214989877848524],
        [83.94695603859866, 28.21528429815527],
        [83.94762817774877, 28.215763204289974],
        [83.94812176892961, 28.216388477490952],
        [83.9483975205341, 28.217110348088877],
        [83.948433478729, 28.217871356524146],
        [83.94822677622989, 28.218610926709633],
        [83.94779386210104, 28.219270187829615],
        [83.94716919380016, 28.219796660777735],
        [83.94640249514741, 28.22014843609342],
        [83.94555479823961, 28.220297510645867],
        [83.945385, 28.22030165978545]]]}
```

```
## Extract the bounding box (bounds) of the buffer ring and form a polygon Geometry.
## i.e. converting circular buffer to poygon (in this case closed square)

bounding_polygon = ee.Geometry.Polygon(buffer_poly.bounds().getInfo()['coordinates'])
bounding_polygon.getInfo()
```

```
{'type': 'Polygon',
 'coordinates': [[[83.94234618509972, 28.214903377646337],
   [83.948433478729, 28.214903377646337],
   [83.948433478729, 28.220301659785473],
   [83.94234618509972, 28.220301659785473],
   [83.94234618509972, 28.214903377646337]]]}
```

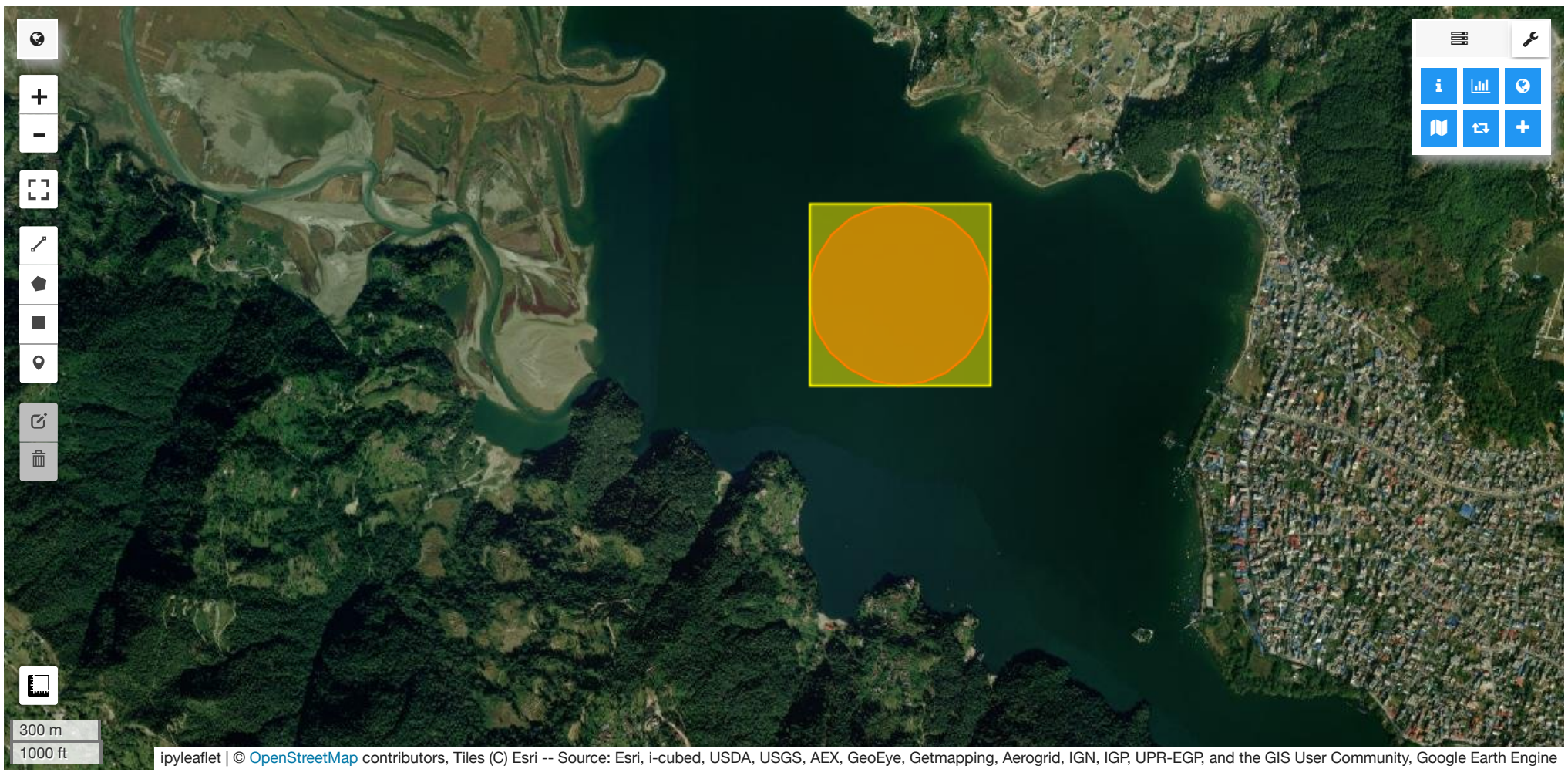
```
## Showing the buffer and bounding polygon

import geemap

## Create a map
my_map = geemap.Map()

# Add geometries to the map
my_map.add_basemap('SATELLITE')
my_map.addLayer(buffer_poly, {'color': 'red'}, "200 meters buffer")
my_map.addLayer(bounding_polygon, {'color': 'yellow'}, "Bounding Polygon")
my_map.centerObject(buffer_poly, zoom=15)
```

```
## Display the map
my_map
```




```
## For a LANDSAT extract the historical temperature data of the lake of interest.

'''
  # Definitions: use these information to choose between different LANDSATs.

    # Paths:

      # `LANDSAT/LT04/C02/T1_L2` --> Landsat 4
      # `LANDSAT/LT05/C02/T1_L2` --> Landsat 5
```

```
      # `LANDSAT/LE07/C02/T1_L2` --> Landsat 7
      # `LANDSAT/LC08/C02/T1_L2` --> Landsat 8
      # `LANDSAT/LC09/C02/T1_L2` --> Landsat 9

    # Coversion requirements:

      # scale = 0.00341802,
      # offset = 149.0,
      # pixel scale = 30

    # Temperature Band Names:

      # Landsat 4 --> ST_B6
      # Landsat 5 --> ST_B6
      # Landsat 7 --> ST_B6
      # Landsat 8 --> ST_B10
      # Landsat 9 --> ST_B10

    # Note: To obtaine most accurate data we need to mask out the pixel which contains clouds, snow, shadows and so on. We can do i
    # bits of a QA_pixel which comes along with each data sample. A detail handwritten note on how to do this is presented in `clou
    # Please refer to that file for detailed explanation.
'''

## Code to generate time series temperature data

# Load Landsat Collection (Surface Reflectance)
landsat_collection = ee.ImageCollection("LANDSAT/LT04/C02/T1_L2") \
.filterBounds(bounding_polygon)         # (Refer definition above)

# Temperature band
temp_band = 'ST_B6'     # (Refer definition above)

# Scale & Offset for band value conversion (Refer definition above)
scale = 0.00341802
offset = 149.0
pix_scale = 30

# Function to create a cloud/snow mask using QA_PIXEL
def cloud_snow_mask(image):

    qa = image.select('QA_PIXEL')  # Pixel Quality Attributes
```

```
    # Extract individual BITMASK values using bitwise operations
    cirrus_confidence = (qa.bitwiseAnd(0b11 << 14)).rightShift(14)  # bits 14-15
    cloud_presence = (qa.bitwiseAnd(0b1 << 3)).rightShift(3)  # bit 3
    cloud_confidence = (qa.bitwiseAnd(0b11 << 8)).rightShift(8)  # bits 8-9
    cloud_shadow_presence = (qa.bitwiseAnd(0b1 << 4)).rightShift(4)  # bit 4
    cloud_shadow_confidence = (qa.bitwiseAnd(0b11 << 10)).rightShift(10)  # bits 10-11
    snow_presence = (qa.bitwiseAnd(0b1 << 5)).rightShift(5)  # bit 5

    # Create masks for presence/absence and confidence values using above bit values
    cirrus_mask = cirrus_confidence.gte(2)  # medium (2) or high (3) confidence
    cloud_mask = cloud_presence.eq(1).And(cloud_confidence.gte(2))  # cloud present AND confidence is medium/high
    shadow_mask = cloud_shadow_presence.eq(1).And(cloud_shadow_confidence.gte(2))  # shadow present AND confidence is medium/high
    snow_mask = snow_presence.eq(1)  # snow presence
```

```
    # Combine all the masks (cloud, shadow, cirrus, snow) using bitwise OR operation
    final_mask = cirrus_mask.Or(cloud_mask).Or(shadow_mask).Or(snow_mask).Not()

    # Apply mask to the ST band images
    masked_image = image.select(temp_band).updateMask(final_mask)

    # Use Scale and Offset to convert image values to temperature
    temp_kelvin = masked_image.multiply(scale).add(offset)

    # Use reducers to compute mean temperature within the bounding polygon
    avg_temp = temp_kelvin.reduceRegion(
        reducer= ee.Reducer.mean(),
        geometry= bounding_polygon,
        scale= pix_scale,
        maxPixels=1e6
    ).get(temp_band)

    # Return temperature data with time stamp
    return image.set('date', image.date().format('YYYY-MM-dd')).set('Mean_Temperature', avg_temp)

# Get the time series temperature data
filtered_collection = landsat_collection.map(cloud_snow_mask)
dates_list = filtered_collection.aggregate_array('date').getInfo()
mean_temps = filtered_collection.aggregate_array('Mean_Temperature').getInfo()
time_series = [(date, temp) for date, temp in zip(dates_list, mean_temps)]

# Print the time series data
```

```
for date, temp in time_series:
    print(f"Date: {date}, Temp: {temp}")
```

```
Date: 1989-01-15, Temp: 288.5044599228912
Date: 1989-01-31, Temp: 289.1273365784906
Date: 1989-03-04, Temp: 292.3648150870348
Date: 1989-01-15, Temp: 288.4178829218017
```

```
# Append the sorted time-series data in `land_temperature_time_series.csv` file

import csv
import numpy as np

data_to_append = time_series  # not sorted
top_path = '/drive'     # Define the top path to the directory to save the file
csv_path = top_path + '/lake_temperature_time_series.csv'


"""

## Important:

# Run the code with this part just once (say while collecting data for just LANDSAT 4 or any LANDSAT from which you wish to start) t
# empty csv file and save the current LANDSAT data. After that comment the code as it is done here.
# Later on the code will just append the data to already saved csv file without any duplication for other LANDSATs.

# create a CSV file
with open(csv_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Date', 'Temperature (K)'])  # header

"""

# Read the existing data from the csv file
with open(csv_path, mode='r') as file:
    reader = csv.reader(file)
    header = next(reader)  # skip header
    prev_data = list(reader)

# Combine current and past data
combined_data = prev_data + data_to_append

# Remove duplicates based on the date, keeping the most recent value
```

```
unique_data = {}
for row in combined_data:
    date, temp = row
    unique_data[date] = temp  # if the date exists, overwrite with the latest temp

# Convert the dictionary back to a sorted list
sorted_combined_data = sorted(unique_data.items(), key=lambda x: x[0])  # sort by date

# Write back to the CSV file
with open(csv_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Date', 'Temperature (K)'])
    writer.writerows(sorted_combined_data)
```

End of Data Collection

Running the above code for each LANDSAT will automatically update the csv file with sorted time-series temperature data. Now we can move to the next part: *data_analysis.ipynb* file.