!pip install rioxarray==0.15.0

!pip install stackstac==0.5.1

!pip install planetary-computer==1.0.0

!pip install odc-stac==0.3.11

!pip install rasterio==1.3.6

!pip install pystac==1.11.0

!pip install geopandas

!pip install shapely

from google.colab import drive

drive.mount('/content/drive', force\_remount = True)

#for casual operations

import warnings

warnings.filterwarnings('ignore')

import pandas as pd

from matplotlib import pyplot as pt

from matplotlib.cm import jet,RdYlGn

import numpy as np

from datetime import datetime

#for the satellites

import stackstac

import pystac\_client

import planetary\_computer

from odc.stac import stac\_load

from shapely.geometry import Polygon

!ls /content/drive/MyDrive/'Colab Notebooks'

FILE\_PATH\_ONE = '/content/drive/MyDrive/Colab Notebooks/Training\_data\_uhi\_index\_UHI2025-v3.csv'

df1 = pd.read\_csv(FILE\_PATH\_ONE) #long and lat, may need more data from daddy Kaggle

df1.head(100)

stac = pystac\_client.Client.open("https://planetarycomputer.microsoft.com/api/stac/v1")

#function to change to appropriate datetime: removal of hours --> year day month to fit retrieval

def datetime\_change(data):

data['datetime'] = data['datetime'][0:10] #removal of the time component

parsed\_datetime = datetime.strptime(data['datetime'], '%d-%m-%Y')

#rearrange the day month year component

rearranged\_date\_str = parsed\_datetime.strftime('%Y-%m-%d')

return str(rearranged\_date\_str)

df1.shape

df1['datetime'].max()

""" Sentinel part """

# Discover and load the data for analysis

# Define the bounding box for the entire data region using (Latitude, Longitude)

# This is the region of New York City that contains our temperature dataset

lower\_left = (40.75, -74.01)

upper\_right = (40.88, -73.86)

# Calculate the bounds for doing an archive data search

# bounds = (min\_lon, min\_lat, max\_lon, max\_lat)

bounds = (lower\_left[1], lower\_left[0], upper\_right[1], upper\_right[0])

# Define the time window

time\_window = "2021-06-01/2021-09-01"

# finding the scenes with overall cloud <30%

stac = pystac\_client.Client.open("https://planetarycomputer.microsoft.com/api/stac/v1")

search = stac.search(

bbox=bounds,

datetime=time\_window,

collections=["sentinel-2-l2a"],

query={"eo:cloud\_cover": {"lt": 10}},

)

# number of scenes that matches the requirement (overall cloud <30%)

items = list(search.get\_items())

print('This is the number of scenes that touch our region:',len(items))

#signed in items

signed\_items = [planetary\_computer.sign(item).to\_dict() for item in items]

# Define the pixel resolution for the final product

# Define the scale according to our selected crs, so we will use degrees

resolution = 30 # meters per pixel

scale = resolution / 111320.0 # degrees per pixel for crs=4326

data = stac\_load(

items,

bands=["B01", "B02", "B03", "B04", "B05", "B06", "B07", "B08", "B8A", "B11", "B12"],

crs="EPSG:4326", # Latitude-Longitude

resolution=scale, # Degrees

chunks={"x": 2048, "y": 2048},

dtype="uint16",

patch\_url=planetary\_computer.sign,

bbox=bounds

)

# View the dimensions of our XARRAY and the loaded variables

# This insures we have the right coordinates and spectral bands in our xarray

# Plot sample images from the time series

import matplotlib.pyplot as plt

plot\_data = data[["B04","B03","B02"]].to\_array()

plot\_data.plot.imshow(col='time', col\_wrap=4, robust=True, vmin=0, vmax=2500)

plt.show()

# Supress Warnings

import warnings

warnings.filterwarnings('ignore')

# Visualization

import matplotlib.pyplot as plt

import seaborn as sns

# Data Science

import numpy as np

import pandas as pd

# Multi-dimensional arrays and datasets

import xarray as xr

# Geospatial raster data handling

import rioxarray as rxr

# Geospatial data analysis

import geopandas as gpd

# Geospatial operations

import rasterio

from rasterio import windows

from rasterio import features

from rasterio import warp

from rasterio.warp import transform\_bounds

from rasterio.windows import from\_bounds

# Image Processing

from PIL import Image

# Coordinate transformations

from pyproj import Proj, Transformer, CRS

# Feature Engineering

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

# Machine Learning

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2\_score

# Planetary Computer Tools

import pystac\_client

import planetary\_computer as pc

from pystac.extensions.eo import EOExtension as eo

# Others

import os

from tqdm import tqdm

"""

Save the output data in a GeoTIFF file

"""

filename = "S2\_sample.tiff"

data\_slice = data.isel(time=2)

# Calculate the dimensions of the file

# height = median.dims["latitude"]

# width = median.dims["longitude"]

height = data\_slice.dims["latitude"]

width = data\_slice.dims["longitude"]

# Define the Coordinate Reference System (CRS) to be common Lat-Lon coordinates

# Define the tranformation using our bounding box so the Lat-Lon information is written to the GeoTIFF

gt = rasterio.transform.from\_bounds(lower\_left[1],lower\_left[0],upper\_right[1],upper\_right[0],width,height)

data\_slice.rio.write\_crs("epsg:4326", inplace=True)

data\_slice.rio.write\_transform(transform=gt, inplace=True);

# Create the GeoTIFF output file using the defined parameters

with rasterio.open(filename,'w',driver='GTiff',width=width,height=height,

crs='epsg:4326',transform=gt,count=11,compress='lzw',dtype='float64') as dst:

dst.write(data\_slice.B01,1)

dst.write(data\_slice.B02,2)

dst.write(data\_slice.B03,3)

dst.write(data\_slice.B04,4)

dst.write(data\_slice.B05,5)

dst.write(data\_slice.B06,6)

dst.write(data\_slice.B07,7)

dst.write(data\_slice.B08,8)

dst.write(data\_slice.B8A,9)

dst.write(data\_slice.B11,10)

dst.write(data\_slice.B12,11)

dst.close()

# Show the location and size of the new output file

!ls \*.tiff

# Reads and plots all bands from the GeoTIFF file.

# Extracts satellite band values from a GeoTIFF based on coordinates from a csv file and returns them in a DataFrame.

def map\_satellite\_data(tiff\_path, csv\_path):

# Load the GeoTIFF data

data = rxr.open\_rasterio(tiff\_path)

tiff\_crs = data.rio.crs

# Read the Excel file using pandas

df = pd.read\_csv(csv\_path)

latitudes = df['Latitude'].values

longitudes = df['Longitude'].values

# 3. Convert lat/long to the GeoTIFF's CRS

# Create a Proj object for EPSG:4326 (WGS84 - lat/long) and the GeoTIFF's CRS

proj\_wgs84 = Proj(init='epsg:4326') # EPSG:4326 is the common lat/long CRS

proj\_tiff = Proj(tiff\_crs)

# Create a transformer object

transformer = Transformer.from\_proj(proj\_wgs84, proj\_tiff)

B01\_values = []

B02\_values = []

B03\_values = []

B04\_values = []

B05\_values = []

B06\_values = []

B07\_values = []

B08\_values = []

B8A\_values = []

B11\_values = []

B12\_values = []

# Iterate over the latitudes and longitudes, and extract the corresponding band values

for lat, lon in tqdm(zip(latitudes, longitudes), total=len(latitudes), desc="Mapping values"):

# Assuming the correct dimensions are 'y' and 'x' (replace these with actual names from data.coords)

B01\_value = data.sel(x=lon, y=lat, band=1, method="nearest").values

B01\_values.append(B01\_value)

B02\_value = data.sel(x=lon, y=lat, band=2, method="nearest").values

B02\_values.append(B02\_value)

B03\_value = data.sel(x=lon, y=lat, band=3, method="nearest").values

B03\_values.append(B03\_value)

B04\_value = data.sel(x=lon, y=lat, band=4, method="nearest").values

B04\_values.append(B04\_value)

B05\_value = data.sel(x=lon, y=lat, band=5, method="nearest").values

B05\_values.append(B05\_value)

B06\_value = data.sel(x=lon, y=lat, band=6, method="nearest").values

B06\_values.append(B06\_value)

B07\_value = data.sel(x=lon, y=lat, band=7, method="nearest").values

B07\_values.append(B07\_value)

B08\_value = data.sel(x=lon, y=lat, band=8, method="nearest").values

B08\_values.append(B08\_value)

B8A\_value = data.sel(x=lon, y=lat, band=9, method="nearest").values

B8A\_values.append(B8A\_value)

B11\_value = data.sel(x=lon, y=lat, band=10, method="nearest").values

B11\_values.append(B11\_value)

B12\_value = data.sel(x=lon, y=lat, band=11, method="nearest").values

B12\_values.append(B12\_value)

# Create a DataFrame with the band values

# Create a DataFrame to store the band values

df = pd.DataFrame()

df['B01'] = B01\_values

df['B02'] = B02\_values

df['B03'] = B03\_values

df['B04'] = B04\_values

df['B05'] = B05\_values

df['B06'] = B06\_values

df['B07'] = B07\_values

df['B08'] = B08\_values

df['B8A'] = B8A\_values

df['B11'] = B11\_values

df['B12'] = B12\_values

return df

# Open the GeoTIFF file

tiff\_path = "S2\_sample.tiff"

# Read the bands from the GeoTIFF file

with rasterio.open(tiff\_path) as src1:

band1 = src1.read(1) # Band [B01]

band2 = src1.read(2) # Band [B02]

band3 = src1.read(3) # Band [B03]

band4 = src1.read(4) # Band [B04]

band5 = src1.read(5) # Band [B05]

band6 = src1.read(6) # Band [B06]

band7 = src1.read(7) # Band [B07]

band8 = src1.read(8) # Band [B08]

band8A = src1.read(9) # Band [B8A]

band11 = src1.read(10) # Band [B11]

band12 = src1.read(11) # Band [B12]

# Mapping satellite data with training data.

final\_data = map\_satellite\_data('S2\_sample.tiff', FILE\_PATH\_ONE)

final\_data.head()

final\_data['NDVI'] = (final\_data['B08'] - final\_data['B04']) / (final\_data['B08'] + final\_data['B04'])

final\_data['NDVI'] = final\_data['NDVI'].replace([np.inf, -np.inf], np.nan)

final\_data['NDBI'] = (final\_data['B11'] - final\_data['B08']) / (final\_data['B11'] + final\_data['B08'])

final\_data['NDBI'] = final\_data['NDBI'].replace([np.inf, -np.inf], np.nan)

final\_data['NDWI'] = (final\_data['B03'] - final\_data['B08']) / (final\_data['B03'] + final\_data['B08'])

final\_data['NDWI'] = final\_data['NDWI'].replace([np.inf, -np.inf], np.nan)

final\_data['GCVI'] = (final\_data['B8A']/final\_data['B03']) - 1

final\_data['GCVI'] = final\_data['GCVI'].replace([np.inf, -np.inf], np.nan)

final\_data['NRDE'] = (final\_data['B8A'] - final\_data['B06']) / (final\_data['B8A'] + final\_data['B06'])

final\_data['NRDE'] = final\_data['NRDE'].replace([np.inf, -np.inf], np.nan)

"""

Joining the predictor variable and response variables

"""

# Combine two datasets vertically (along columns) using pandas concat function.

def combine\_two\_datasets(dataset1,dataset2):

'''

Returns a vertically concatenated dataset.

Attributes:

dataset1 - Dataset 1 to be combined

dataset2 - Dataset 2 to be combined

'''

data = pd.concat([dataset1,dataset2], axis=1)

return data

# Combining ground data and final data into a single dataset.

uhi\_data = combine\_two\_datasets(df1,final\_data)

uhi\_data.head()

# Resetting the index of the dataset

uhi\_data=uhi\_data.reset\_index(drop=True)

display(uhi\_data)

"""

Model Building

"""

# Retaining only the columns for B01, B06, NDVI, and UHI Index in the dataset.

uhi\_data = uhi\_data[['NDVI', 'NDBI', 'NDWI', 'GCVI', 'NRDE','B11', 'B12', 'B8A', 'B05', 'B04','UHI Index']]

display(uhi\_data)

"""

Train and Test Split

"""

# Split the data into features (X) and target (y), and then into training and testing sets

X = uhi\_data.drop(columns=['UHI Index']).values

y = uhi\_data ['UHI Index'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3,random\_state=123)

# Scale the training and test data using standardscaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

"""

Model Training

"""

# Train the Random Forest model on the training data

model = RandomForestRegressor(n\_estimators=200, random\_state=42)

model.fit(X\_train,y\_train)

"""

Model Evaluation

"""

# Make predictions on the training data

# insample\_prediction

insample\_predictions = model.predict(X\_train)

# calculate R-squared score for in-sample predictions

Y\_train = y\_train.tolist()

r2\_score(Y\_train, insample\_predictions)

# outsampl\_prediction

# Make predictions on the test data

outsample\_predictions = model.predict(X\_test)

# calculate R-squared score for out-sample predictions

Y\_test = y\_test.tolist()

r2\_score(Y\_test, outsample\_predictions)

"""

Submission

"""

Submission\_template = '/content/drive/MyDrive/Colab Notebooks/Submission\_template\_UHI2025-v3.csv'

#Reading the coordinates for the submission

test\_file = pd.read\_csv(Submission\_template)

test\_file.head()

# Mapping satellite data for submission.

val\_data = map\_satellite\_data('S2\_sample.tiff',Submission\_template )

# Calculate NDVI (Normalized Difference Vegetation Index) and handle division by zero by replacing infinities with NaN.

val\_data['NDVI'] = (val\_data['B08'] - val\_data['B04']) / (val\_data['B08'] + val\_data['B04'])

val\_data['NDVI'] = val\_data['NDVI'].replace([np.inf, -np.inf], np.nan) # Replace infinities with NaN

# Calculate NDBI (Normalized Difference Buildup Index) and handle division by zero by replacing infinities with NaN

val\_data['NDBI'] = (val\_data['B11'] - val\_data['B08']) / (val\_data['B11'] + val\_data['B08'])

val\_data['NDBI'] = val\_data['NDBI'].replace([np.inf, -np.inf], np.nan)

# Calculate NDWI (Normalized Difference Water Index) and handle division by zero by replacing infinities with NaN

val\_data['NDWI'] = (val\_data['B03'] - val\_data['B08']) / (val\_data['B03'] + val\_data['B08'])

val\_data['NDWI'] = val\_data['NDWI'].replace([np.inf, -np.inf], np.nan)

val\_data['GCVI'] = (val\_data['B8A']/val\_data['B03']) - 1

val\_data['GCVI'] = val\_data['GCVI'].replace([np.inf, -np.inf], np.nan)

val\_data['NRDE'] = (val\_data['B8A'] - val\_data['B06']) / (val\_data['B8A'] + val\_data['B06'])

val\_data['NRDE'] = val\_data['NRDE'].replace([np.inf, -np.inf], np.nan)

val\_data.head()

# Extracting specific columns (B01, B06, and NDVI) from the validation dataset

submission\_val\_data=val\_data.loc[:,['NDVI', 'NDBI', 'NDWI', 'GCVI', 'NRDE','B11', 'B12', 'B8A', 'B05', 'B04']]

submission\_val\_data.head()

# Feature Scaling

submission\_val\_data = submission\_val\_data.values

transformed\_submission\_data = sc.transform(submission\_val\_data)

#Making predictions

final\_predictions = model.predict(transformed\_submission\_data)

final\_prediction\_series = pd.Series(final\_predictions)

#Combining the results into dataframe

submission\_df = pd.DataFrame({'Longitude':test\_file['Longitude'].values, 'Latitude':test\_file['Latitude'].values, 'UHI Index':final\_prediction\_series.values})

#Displaying the sample submission dataframe

submission\_df.head()

#Dumping the predictions into a csv file.

#finish for the modeling

file\_path = '/content/drive/MyDrive/Colab Notebooks/submission\_df\_trial5.csv'

submission\_df.to\_csv(file\_path, index = False)

How do you implement the other logic to rewrite it to make it more efficient and more accurate, aim to have r-score of 0.9

1. Grab the lat long from the submission template file, run it thru stac\_load for landsat and sentinel
2. Grab the bands we need, can refer to prev cells
3. Then arrange become a feature dataset(exclude lat long) to be put thru the pred model
4. Grab the Uhi index from the model and put it with the lat long submission file temp
5. Then u get a complete csv to be uploaded