Remote Sensing and NDVI Timeseries

Understanding NDVI theory:

NDVI: Normalized Difference Vegetation Index. It is a vegetation index that measures/represents the greenness of an area. From the greenness we can infer the quantity and quality of the vegetation in that area. Say, high NDVI of a agricultural area means healthy crops.

$NDVI = \frac{NIR-R}{NIR+R}$; value ranges from -1 to +1, base soil's NDVI = 0, water, snow, cloud= negative value.

NIR = Near infrared radiation band (0.7-1.1 μm) Green plants reflect it

R = Red band (0.6 - 0.7 μm) plants absorb it.

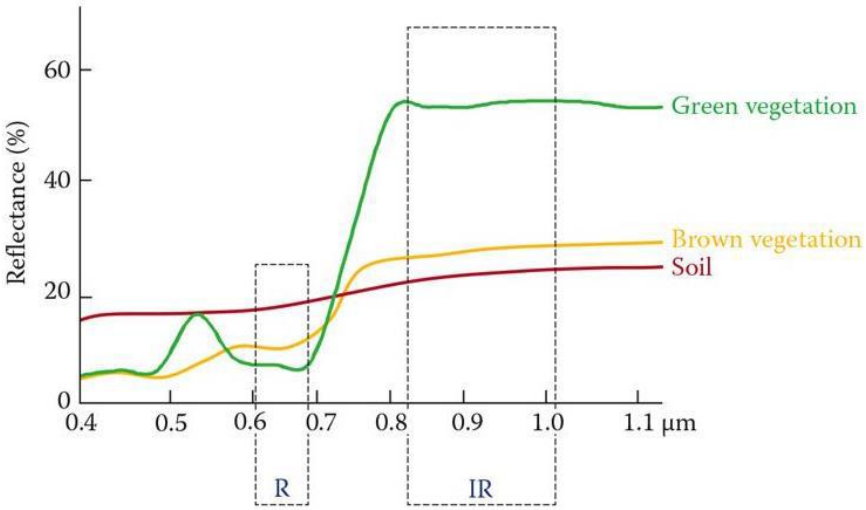

Figure 1: Spectral signature pf materials (collected from web)

Limitation of NDVI index: Sometimes soil brightness impacts NDVI score. The same vegetation gives higher NDVI if the soil is dark or wet.

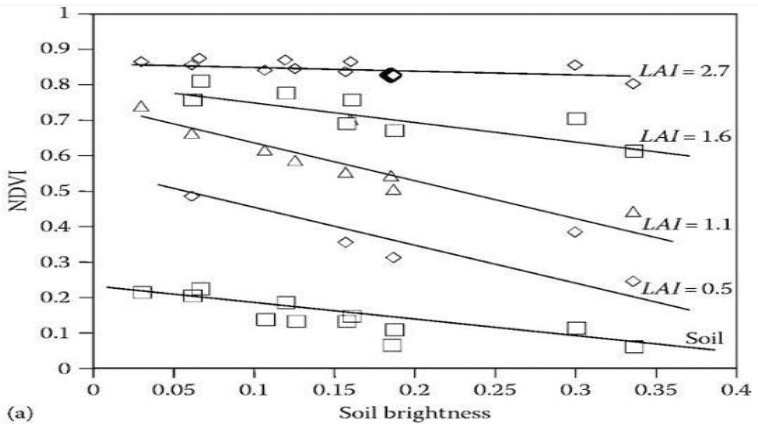

Figure 2: Soil brightness affecting NDVI score (collected from web)

Some applications of NDVI:

Domain	Application
Hydrology	- Vegetation controls **evapotranspiration**, **soil moisture**, and **runoff**. - Seasonal NDVI trends show land-surface processes. - Forest cover and changes can impact watershed behavior.
Agriculture	- Crop monitoring (growth stages, health). - Predict yield. - Detect stress (drought, pests, disease).
Climate Studies	- Observe ecosystem responses to climate change. - Drought monitoring (low NDVI = drought stress).
Urban Planning	- Measure urban green spaces. - Detect urban heat islands (correlate low NDVI with higher land surface temperature).
Environmental Monitoring	- Forest loss/gain detection. - Desertification studies. - Fire impact assessment.
Disaster Management	- Floods and drought monitoring (using vegetation stress signs).

Set up google earth engine map (geemap)

GEE is a massive geospatial cloud platform that provides a huge repository of satellite imagery and geospatial datasets. To get GEE running in python we have to use the package ee. Addition to that, we use geemap that gives us more functions and tools for analyzing GEE datasets better. Capabilities of GEE:

Feature	What it allows you to do
Interactive Mapping	Add, remove, style layers dynamically inside Jupyter
Data Access	Load huge GEE datasets easily (Sentinel, MODIS, Landsat)
Visualization	Apply color palettes, thresholds, time sliders to imagery
Drawing Tools	Extract mean/median values from imagery over shapefiles
Zonal Stats	Extract mean/median values from imagery over shapefiles
Export Tools	Export GEE results (e.g., clipped rasters) to your drive
Animation	Create simple GIFs (e.g., NDVI time series over time)

- We can check geemap documentation here: geemap.
- Some geemap youtube tutorials here: (134) Open Geospatial Solutions - YouTube
- Some Pyhton notebook examples of using geemap here: https://github.com/giswqs/earthengine-py-notebooks
- The geemap book which also works like a user manual: Earth Engine and Geemap — Geospatial Data Science with Earth Engine and Geemap

Earth Engine is a cloud service. To use this service, we need a GEE account. While initializing the geemap, python will ask permission to access your cloud service using our account. We need to permit it. Easy steps to follow:

- Log in to your **GEE account**.
- Allow permissions.
- Paste a token or automatically authenticate.

Accessing the sentinel / MODIS dataset:

Before getting into how to access the data, we can go through some basic information to put things into perspective.

GEE stores data in the following data types:

- ee.Image – Think it as a raster
- ee.ImageCollection – A timeseries of rasters
- ee.Geometry – A basic shape, think it as a bounding box. Often used as spatial filter
- ee.Feature – A geometry+attribute data. Think of it as an element in the shapefile
- ee.FeatureCollection – think of it as a shape file or GeoJSON file

The earth engine data catalog: Earth Engine Data Catalog | Google for Developers gives an overview of the data available. For NDVI data we can use sources like MODIS, Landsat, or sentinel where the first two are NASA products and the later one is Copernicus(EU) product. The selection criteria mostly depend on the temporal and spatial resolution of the data and our requirements.

However, no satellite directly measures NDVI. So, there is no data product giving NDVI measurements (Expect in some cases some preprocessed products are available, like MODIS). We are actually looking for surface reflectance data and the RED and NEAR INFRARED bands in it.

We decided to go with Landsat data which has a 16-day interval, 30m resolution: ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")

Interpreting some information embedded in the dataset:

C02 = Collection 02 = Latest calibration

T1 = Tier 1 = Highest quality scenes (Geometrically and radiometrically sound)

L2 = Level 2 = Surface reflectance atmospherically corrected

Band Information provided in data source:

Name	Units	Min	Max	Scale	Offset	Wavelength	Description
SR_B1		1	65455	2.75e-05	-0.2	0.435-0.451 µm	Band 1 (ultra blue, coastal aerosol) surface reflectance
SR_B2		1	65455	2.75e-05	-0.2	0.452-0.512 µm	Band 2 (blue) surface reflectance
SR_B3		1	65455	2.75e-05	-0.2	0.533-0.590 µm	Band 3 (green) surface reflectance
SR_B4		1	65455	2.75e-05	-0.2	0.636-0.673 µm	Band 4 (red) surface reflectance
SR_B5		1	65455	2.75e-05	-0.2	0.851-0.879 µm	Band 5 (near infrared) surface reflectance
SR_B6		1	65455	2.75e-05	-0.2	1.566-1.651 µm	Band 6 (shortwave infrared 1) surface reflectance
SR_B7		1	65455	2.75e-05	-0.2	2.107-2.294 µm	Band 7 (shortwave infrared 2) surface reflectance

We want to use the SR_B4 and SR_B5 bands. What is important is the scale and offset information. The raster data is stored as integer and to get the real reflectance value, we need to multiply it with scale and add the offset.

As we want to produce an NDVI timeseries at the end, we need NDVI data over a period of time. Means we need a ImageCollection. We access the dataset using ee.ImageCollection() function that gives an image collection object. Then, Filter it out to our area of interest (aoi) and the time range. To filter we need to create an aoi geometry beforehand using the ee.Geometry.BBox() function. Then, we can select the desired bands that we need.

Clipping to Watershed: We only need this step as we access the data using a bounding box around our watershed.

To clip according to our watershed, we need a geometry of our watershed. And we already have a geometry of our watershed when we loaded the shapefile using geopandas. But unfortunately, ee package does not understand geopandas geometry (shapely format), it only understands GeoJSON geometry. Thus, we need to convert our geopandas(shapely) geometry to GeoJSON geometry. Some functions:

Cloud Masking: Cloud masking is a product specific process, which means we do it differently for Landsat, sentinel, MODIS. The Landsat data in my opinion has the easiest and clearest masking process. Each Landsat 8 Level-2 image contains a band called "QA_PIXEL". This band

holds **bit-packed quality information** — like a tag on every pixel saying: "This pixel is cloudy" / "This pixel has snow" / "This is a good pixel", etc.

Learn detail about cloud masking here: Cloud masking on GEE using Landsat 8 | Life in GIS

Calculating NDVI: GEE images have built in attribute(function) called normalizedDifference() to calculate normalized difference between different bands. As NDVI is the normalized difference between NRI and RED band, we can use this function to calculate NDVI. The function returns a single band raster with NDVI which we can add to our original raster image using addBands() function.

Visualizing NDVI: In case of GEE, visualization is done using geemap

If NDVI is a timeseries, we need to select one single image to visualize by first() (select the first image to visualize). But for remote sensing data selecting one image is not a good option as the selected image might lose lot of data due to cloud masking. Thus, we create composites and visualize. Example: mean composite – calculates the mean NDVI from all the images over the whole timeline.

Investigating NDVI Raster Value:

To check if the NDVI values are sensible we can check some stats like the max/mean value, the average value. We use the reduceRegion() function.

I) Checking max, min, mean statistics
II) Pixel Counts

To investigate we can use the reduceRegion() attribute of the image object. And we can count the pixel that has valid value.

Visual Beautification:

1. Custom color palette (More colors, sharper map)
2. Calibrated min/max range
3. Legend
4. Opacity and Boundary overlay

Extracting NDVI timeseries using zonal statistics: To build a timeseries of NDVI, so that the seasonal variation can be examined, we need to extract the mean NDVI value from each image. We will use the same reduceRegion() function to calculate the mean but this time all the images will go through this function and ultimately create a NDVI timeseries and save it as a pandas dataframe.

Handling missing data in NDVI timeseries: In the NDVI timeseries there might be some missing values. To tackle the missing values, we can use different interpolation methods depending on the length of the gap, seasonality etc. Here, we tried two interpolation methods: Spline and linear interpolation and the later one came out better filling the gaps.