Sentinel 1 and World Cover

Important Libraries

```
import ee
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from datetime import datetime
from datetime import timedelta
!pip install geemap
ee.Authenticate()
```

```
Collecting geemap
  Downloading geemap-0.24.1-py2.py3-none-any.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 53.0 MB/s eta 0:00:00
Collecting bqplot (from geemap)
  Downloading bqplot-0.12.39-py2.py3-none-any.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 47.3 MB/s eta 0:00:00
Collecting colour (from geemap)
  Downloading colour-0.1.5-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: earthengine-api>=0.1.347 in /usr/local/lib/python3.10/dist-packages (from geemap) (0.1.357)
Collecting eerepr>=0.0.4 (from geemap)
  Downloading eerepr-0.0.4-py3-none-any.whl (9.7 kB)
Requirement already satisfied: folium>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from geemap) (0.14.0)
Collecting geocoder (from geemap)
  Downloading geocoder-1.38.1-py2.py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.6/98.6 kB 12.0 MB/s eta 0:00:00
Collecting ipyevents (from geemap)
  Downloading ipyevents-2.0.1-py2.py3-none-any.whl (130 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 130.5/130.5 kB 12.6 MB/s eta 0:00:00
Collecting ipyfilechooser>=0.6.0 (from geemap)
  Downloading ipyfilechooser-0.6.0-py3-none-any.whl (11 kB)
Collecting ipyleaflet>=0.17.0 (from geemap)
  Downloading ipyleaflet-0.17.3-py3-none-any.whl (3.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 53.5 MB/s eta 0:00:00
Collecting ipytree (from geemap)
  Downloading ipytree-0.2.2-py2.py3-none-any.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 48.6 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from geemap) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from geemap) (1.22.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from geemap) (1.5.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from geemap) (5.13.1)
Collecting pyperclip (from geemap)
  Downloading pyperclip-1.8.2.tar.gz (20 kB)
  Preparing metadata (setup.py) ... done
Collecting pyshp>=2.1.3 (from geemap)
  Downloading pyshp-2.3.1-py2.py3-none-any.whl (46 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.5/46.5 kB 5.6 MB/s eta 0:00:00
Collecting python-box (from geemap)
  Downloading python_box-7.0.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 82.0 MB/s eta 0:00:00
Collecting scooby (from geemap)
  Downloading scooby-0.7.2-py3-none-any.whl (16 kB)
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (2.8.0)
Requirement already satisfied: google-api-python-client>=1.12.1 in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (2.84.0)
Requirement already satisfied: google-auth>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (2.17.3)
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (0.1.0)
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (0.21.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from earthengine-api>=0.1.347->geemap) (2.27.1)
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from folium>=0.13.0->geemap) (0.6.0)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages (from folium>=0.13.0->geemap) (3.1.2)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from ipyfilechooser>=0.6.0->geemap) (7.7.1)
Collecting traittypes<3,>=0.2.1 (from ipyleaflet>=0.17.0->geemap)
  Downloading traittypes-0.2.1-py2.py3-none-any.whl (8.6 kB)
Collecting xyzservices>=2021.8.1 (from ipyleaflet>=0.17.0->geemap)
  Downloading xyzservices-2023.5.0-py3-none-any.whl (56 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 kB 7.7 MB/s eta 0:00:00
Requirement already satisfied: traitlets>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from bqplot->geemap) (5.7.1)
```

```
ee.Initialize()
import geemap
from geemap.plot import center_zoom_to_xy_range
import ipywidgets as widgets
from ipyleaflet import WidgetControl
from geemap import geojson_to_ee
```

Region of Interest

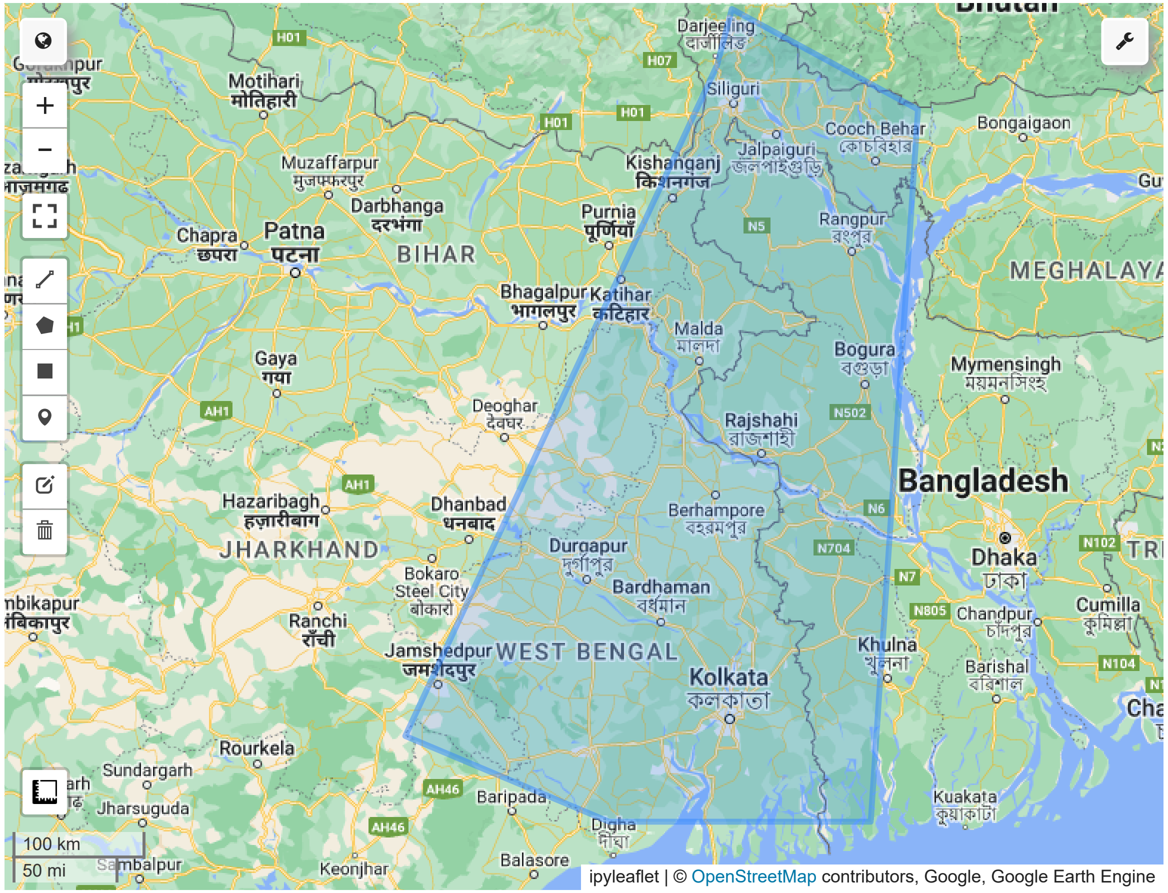
```
import geemap
from geemap.plot import center_zoom_to_xy_range
import ipywidgets as widgets
from ipyleaflet import WidgetControl
from geemap import geojson_to_ee

Map1 = geemap.Map(center =[23.8402, 87.6186], zoom_start=50)

dc = Map1.draw_control

roi = []
# Handle draw events
def handle_draw(self, action, geo_json):
    geometry = geo_json['geometry']
    if geometry['type'] == 'Polygon':
        coordinates = geometry['coordinates'][0]
        roi.append(coordinates)
        print("Polygon coordinates:", coordinates)
```

```
dc.on_draw(handle_draw)
Map1
```




```
Polygon coordinates: [[88.280484, 26.980829], [85.952222, 22.451649], [87.467789, 21.861499], [89.42265
```

Rice Group Selection By hand on Map

```
polygon = ee.Geometry.Polygon(roi)

Map = geemap.Map(center =[23.8402, 87.6186], zoom_start=9)


sentinel1 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-06-01', '2022-06-15').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel2 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-06-16', '2022-06-30').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel3 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-07-01', '2022-07-15').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel4 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-07-16', '2022-07-31').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
```

```
sentinel5 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-08-01', '2022-08-15').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel6 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-08-16', '2022-08-31').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel7 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-09-01', '2022-09-15').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel8 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-09-16', '2022-09-30').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel9 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-10-01', '2022-10-15').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.e
sentinel10 = ee.ImageCollection('COPERNICUS/S1_GRD').filterDate('2022-10-16', '2022-10-31').filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH')).filter(ee.Filter.


image1 = sentinel1.select('VH').mean().rename('VH1')
image2 = sentinel2.select('VH').mean().rename('VH2')
image3 = sentinel3.select('VH').mean().rename('VH3')
image4 = sentinel4.select('VH').mean().rename('VH4')
image5 = sentinel5.select('VH').mean().rename('VH5')
image6 = sentinel6.select('VH').mean().rename('VH6')
image7 = sentinel7.select('VH').mean().rename('VH7')
image8 = sentinel8.select('VH').mean().rename('VH8')
image9 = sentinel9.select('VH').mean().rename('VH9')
image10 = sentinel10.select('VH').mean().rename('VH10')

stacked = image1.addBands([image2,image3,image4,image5,image6,image7,image8,image9,image10]).clip(polygon)

stacked_scaled = stacked.multiply(10).add(350).uint8();
bands = ['VH7','VH8','VH9']
display = {'bands': bands,'min': 0, 'max': 220}





# Load the WorldCover dataset
dataset = ee.ImageCollection("ESA/WorldCover/v100").first().clip(polygon)
# Update the dataset to only include agricultural land (class 40)
dataset_agri = dataset.updateMask(dataset.eq(40))
# Visualization parameters
visualization = {
  'bands': 'Map'
}
# Center the map on the dataset
Map.centerObject(dataset)




# Add the landcover layer to the map
Map.addLayer(dataset, visualization, "Landcover")

# Add the stacked layer to the map
Map.addLayer(stacked_scaled, display, 'stacked')

# Get the DrawControl
dc = Map.draw_control
```

```
# List of recognised Fields
polygon_coordinates = []

# Handle draw events
def handle_draw(self, action, geo_json):
    geometry = geo_json['geometry']
    if geometry['type'] == 'Polygon':
        coordinates = geometry['coordinates'][0]
        polygon_coordinates.append(coordinates)
        print("Polygon coordinates:", coordinates)

dc.on_draw(handle_draw)


Map
# print(image1)
```


```
polygon_coordinates
```

```
[[[88.157623, 22.013087],
  [88.119862, 21.97425],
  [88.166548, 21.978071],
  [88.157623, 22.013087]],
 [[88.52661, 24.367426],
  [88.517513, 24.3582],
  [88.531072, 24.355073],
  [88.52661, 24.367426]],
 [[86.018645, 23.040877],
  [86.010922, 23.01876],
  [86.032548, 23.029503],
  [86.018645, 23.040877]],
 [[86.743054, 22.984313],
  [86.74786, 22.966138],
  [86.76571, 22.976886],
  [86.743054, 22.984313]],
 [[87.949316, 22.299103],
  [87.937473, 22.282743],
  [87.954465, 22.282743],
  [87.949316, 22.299103]]]
```

For Generating Dataset


```
# function to get the rice fields of requirement
def generate_fields (polygon_coordinates):
  fields = []
  for field in polygon_coordinates :
    field = np.array(field)
    field = np.transpose(field)
    for i in range(10):
      coeff = np.random.rand(field.shape[1])
      coeff /= coeff.sum()
      fieldpoint = np.dot(field, coeff)
      fields.append(list(fieldpoint))
  return fields
```

```
# Getting the rice fields
fields = generate_fields(polygon_coordinates) #Here the polygon_coordinates contains the primary rice fields
fields
```

```
[[88.14929901695433, 21.999820717686642],
 [88.14878089004156, 22.001664574307046],
 [88.1379142757752, 21.985370193217243],
```

```
 [88.15355821134824, 22.001968710532836],
 [88.15497512800562, 21.994677637316258],
 [88.15966185857164, 22.00466679425826],
 [88.15505989111284, 21.99641094895167],
 [88.13989318605311, 21.993776778277226],
 [88.14923905236577, 21.984664897798417],
 [88.14833603059483, 21.998677726693675],
 [88.52601207542216, 24.359200590564637],
 [88.52325838969416, 24.36160573466132],
 [88.52651745819094, 24.364783803414667],
 [88.52351733362516, 24.363259868693113],
 [88.52433619289164, 24.363879573507013],
 [88.52641843838441, 24.36235332429564],
 [88.52713327191111, 24.3611216244194 6],
 [88.52576195188234, 24.362386808277574],
 [88.52661948615925, 24.360327992495023],
 [88.52787984853384, 24.36362638417426],
 [86.02117336778245, 23.03054372083485],
 [86.02113646989844, 23.030362937122508],
 [86.02258730220447, 23.032513412200124],
 [86.02369743928978, 23.033822622941905],
 [86.019754288256, 23.03080396505268],
 [86.01902008513021, 23.03657510248018],
 [86.0202533373218, 23.035557534320425],
 [86.01996868734982, 23.029558428235312],
 [86.02074572843685, 23.033143706184447],
 [86.01945867771288, 23.028317531119143],
 [86.74986157354928, 22.980902726104894],
 [86.75235479488046, 22.97444515508009],
 [86.74590492132545, 22.97512277760144],
 [86.75413345396171, 22.97106629516291],
 [86.75578996355324, 22.97447765685429],
 [86.74740526009569, 22.977211121136378],
 [86.74519256885364, 22.978439264557224],
 [86.7473239506646, 22.978585787900293],
 [86.7499743992669, 22.979566782062395],
 [86.74607336361228, 22.97606922572744],
 [87.9494411520029, 22.2905653334806],
 [87.94383466643357, 22.29115246173332],
 [87.94982892805203, 22.284963704177954],
 [87.94804677976374, 22.289647531308287],
 [87.94794832170024, 22.29241088509951],
 [87.94590069677261, 22.29132906657988],
 [87.95007750578407, 22.290232921625154],
 [87.9429473137304, 22.28676613687001],
 [87.94651894242214, 22.291747514023193],
 [87.9484538643, 22.292995304054553]]
```

```
# Making the dataframe which will contain our training dataset
df = pd.DataFrame(np.array(fields)).rename({0 : 'latitude', 1 : 'longitude'}, axis = 1)
df.shape
```

```
(50, 2)
```

```
# Adding Rice-Groups Class
df['Rice-Groups']=None
```

```
for i in range(1,11):
  ls = []
  for j in range(df.shape[0]):
    pointOfInterest = ee.Geometry.Point([df.iloc[j][0],df.iloc[j][1]])
    bandValues = stacked_scaled.reduceRegion(
      reducer = ee.Reducer.first(),  # You can choose a different reducer if needed
      geometry = pointOfInterest,
      scale = 30,  # Specify the scale/resolution for the analysis
      maxPixels = 30  # Set a limit for the number of pixels to be processed
    )
    ls.append(ee.Number(bandValues.get('VH'+str(i))).toInt().getInfo())
  df['VH'+str(i)] = pd.Series(np.array(ls))
  df['Rice-Groups']='Waterbody'
```

```
df
```

	latitude	longitude	Rice-Groups	VH1	VH2	VH3	VH4	VH5	VH6	VH7	VH8	VH9	VH10
0	88.149299	21.999821	Waterbody	81	88	87	90	80	87	87	67	105	98
1	88.148781	22.001665	Waterbody	82	88	78	78	71	75	76	105	76	117
2	88.137914	21.985370	Waterbody	85	75	75	82	82	66	64	61	81	99
3	88.153558	22.001969	Waterbody	87	82	100	107	82	92	69	76	80	101
4	88.154975	21.994678	Waterbody	78	90	101	64	89	83	87	81	80	131
5	88.159662	22.004667	Waterbody	90	94	77	82	85	91	91	80	88	106
6	88.155060	21.996411	Waterbody	62	79	86	74	82	87	74	90	82	127
7	88.139893	21.993777	Waterbody	77	81	78	73	83	79	83	74	88	110
8	88.149239	21.984665	Waterbody	73	75	85	64	87	76	93	106	99	102
9	88.148336	21.998678	Waterbody	85	76	94	86	69	87	78	66	85	101
10	88.526012	24.359201	Waterbody	73	90	71	89	74	69	77	101	72	98
11	88.523258	24.361606	Waterbody	69	93	87	83	89	84	66	84	75	109
12	88.526517	24.364784	Waterbody	75	86	92	81	78	75	83	77	89	97
13	88.523517	24.363260	Waterbody	85	99	74	81	90	89	84	82	92	119
14	88.524336	24.363880	Waterbody	80	84	76	77	84	81	72	77	64	89
15	88.526418	24.362353	Waterbody	87	91	92	69	81	65	73	74	82	112
16	88.527133	24.361122	Waterbody	78	74	70	60	82	86	60	75	65	103
17	88.525762	24.362387	Waterbody	76	75	85	80	86	84	97	67	64	107
18	88.526619	24.360328	Waterbody	91	75	104	95	62	73	75	79	88	116
19	88.527880	24.363626	Waterbody	67	84	80	82	68	70	72	65	82	105
20	86.021173	23.030544	Waterbody	97	92	96	88	81	89	92	96	78	78
21	86.021136	23.030363	Waterbody	88	95	98	86	98	85	87	94	82	93
22	86.022587	23.032513	Waterbody	92	86	84	88	91	93	89	96	106	81
23	86.023697	23.033823	Waterbody	101	79	85	83	74	93	70	82	100	86
24	86.019754	23.030804	Waterbody	81	88	94	69	96	101	84	83	84	73
25	86.019020	23.036575	Waterbody	89	93	94	71	115	97	81	64	97	71
26	86.020253	23.035558	Waterbody	92	95	83	80	80	82	76	83	87	79
27	86.019969	23.029558	Waterbody	86	93	91	95	89	92	76	74	96	76
28	86.020746	23.033144	Waterbody	86	85	86	78	83	99	88	85	93	100
29	86.019459	23.028318	Waterbody	78	93	95	80	88	92	90	99	99	94

30	86.749862	22.980903	Waterbody	71	68	83	80	52	68	78	77	69	82
31	86.752355	22.974445	Waterbody	60	76	76	84	65	61	85	66	78	77
32	86.745905	22.975123	Waterbody	93	67	70	83	73	74	77	85	72	74

```
# Concatenate the DataFrames vertically
df_final = pd.concat([df_final,df])

# # Reset the index of the concatenated DataFrame
df_final.reset_index(drop=True, inplace=True)
```

```
df_final
```

	latitude	longitude	Rice-Groups	VH1	VH2	VH3	VH4	VH5	VH6	VH7	VH8	VH9	VH10
0	87.769473	24.303725	Class1	153	131	84	129	134	162	180	188	214	201
1	87.769445	24.303673	Class1	153	131	84	129	134	162	180	188	214	201
2	87.769436	24.303653	Class1	153	131	84	129	134	162	180	188	214	201
3	87.769434	24.303657	Class1	153	131	84	129	134	162	180	188	214	201
4	87.769395	24.303627	Class1	153	131	84	129	134	162	180	188	214	201
...	...	...	...	...	...	...	...	...	...	...	...	...	...
405	87.945901	22.291329	Waterbody	86	95	83	83	91	81	79	80	77	100
406	87.950078	22.290233	Waterbody	89	102	85	91	69	84	88	78	85	98
407	87.942947	22.286766	Waterbody	81	85	92	100	84	79	96	90	84	85
408	87.946519	22.291748	Waterbody	75	76	78	89	74	91	85	91	86	102
409	87.948454	22.292995	Waterbody	86	85	96	103	78	100	74	107	93	94

410 rows × 13 columns

```
df_final
```

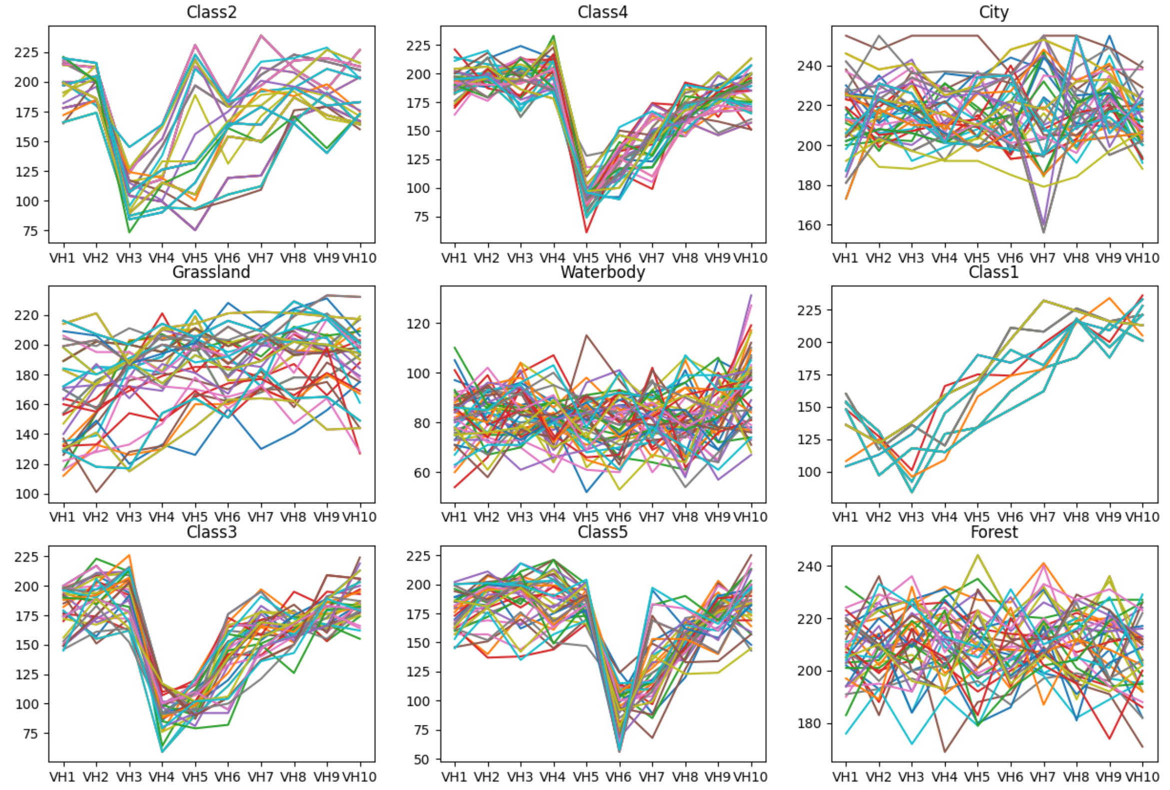
	latitude	longitude	Rice-Groups	VH1	VH2	VH3	VH4	VH5	VH6	VH7	VH8	VH9	VH10
0	87.769473	24.303725	Class1	153	131	84	129	134	162	180	188	214	201
1	87.769445	24.303673	Class1	153	131	84	129	134	162	180	188	214	201
2	87.769436	24.303653	Class1	153	131	84	129	134	162	180	188	214	201
3	87.769434	24.303657	Class1	153	131	84	129	134	162	180	188	214	201
4	87.769395	24.303627	Class1	153	131	84	129	134	162	180	188	214	201

```
ds = df_final['Rice-Groups'].value_counts()
ds
```

```
Class2       50
Class4       50
City         50
Grassland    50
Waterbody    50
Class1       40
Class3       40
Class5       40
Forest       40
Name: Rice-Groups, dtype: int64
```

```
df1 = df_final
```

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize']=(15,10)
figure, ax = plt.subplots(3,3)
k =0
for j in list(ds.index):
  for i in range(int(df1[df1["Rice-Groups"] == j ].shape[0])):
    ax[int(k/3)][k%3].plot((df1[df1["Rice-Groups"] == j].iloc[i])[3:])
  ax[int(k/3)][k%3].title.set_text(str(j))
  k = k+1
```




```
df1.to_csv('newdata.csv')
```

Clustering

```
df = pd.read_csv('newdata.csv',)
```

```
df.drop('Unnamed: 0', inplace=True, axis=1)
```

```
df.head()
```

	latitude	longitude	Rice-Groups	VH1	VH2	VH3	VH4	VH5	VH6	VH7	VH8	VH9	VH10
0	87.769473	24.303725	Class1	153	131	84	129	134	162	180	188	214	201
1	87.769445	24.303673	Class1	153	131	84	129	134	162	180	188	214	201
2	87.769436	24.303653	Class1	153	131	84	129	134	162	180	188	214	201

```
df['Rice-Groups'].value_counts()
```

```
Class2       50
Class4       50
City         50
Grassland    50
Waterbody    50
Class1       40
Class3       40
Class5       40
Forest       40
Name: Rice-Groups, dtype: int64
```

```
from sklearn.cluster import *
from sklearn.preprocessing import StandardScaler
```

```
X = df.loc[:,'VH1':]
```

```
clf1 = KMeans(n_clusters = 9)
clf2 = AgglomerativeClustering(n_clusters = 9)
clf3 = DBSCAN(eps = 0.4, min_samples = 5)
```

```
clf1.fit(X)
clf2.fit(X)
clf3.fit(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default valu
  warnings.warn(
▾ DBSCAN
DBSCAN(eps=0.4)
```

```
df["y_pred_means"] = clf1.fit_predict(X)
df["y_pred_hierarchial"] = clf2.fit_predict(X)
df["y_pred_density"] = clf3.fit_predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_i
  warnings.warn(
```

```
ls = []
for i in list(df["Rice-Groups"].unique()):
  dict = {"means":[],"hierarchial":[],"density":[]}
  dict["means"] = list(df[df["Rice-Groups"] == i]["y_pred_means"].unique())
  dict["hierarchial"] = list(df[df["Rice-Groups"] == i]["y_pred_hierarchial"].unique())
  dict["density"] = list(df[df["Rice-Groups"] == i]["y_pred_density"].unique())
  dict = {i:dict}
  ls.append(dict)
```

```
ls
```

```
[{'Class1': {'means': [3], 'hierarchial': [0], 'density': [0, 1, -1, 2, 3]}},
 {'Class2': {'means': [4, 5],
   'hierarchial': [8, 1],
   'density': [4, -1, 5, 6, 7]}},
 {'Class3': {'means': [6], 'hierarchial': [3], 'density': [-1]}},
 {'Class4': {'means': [1], 'hierarchial': [6], 'density': [-1, 8]}},
 {'Class5': {'means': [8], 'hierarchial': [2], 'density': [-1]}},
 {'Forest': {'means': [7, 2], 'hierarchial': [4, 5], 'density': [-1]}},
 {'City': {'means': [2, 7], 'hierarchial': [5, 4], 'density': [-1]}},
 {'Grassland': {'means': [7, 3, 2, 5],
   'hierarchial': [4, 5, 0],
   'density': [-1]}},
 {'Waterbody': {'means': [0], 'hierarchial': [7], 'density': [-1]}}]
```

Training Algorithm

```
# Machine Learning Model : Random Forest Classifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
X = df.drop('Rice-Groups',axis=1)
y = df['Rice-Groups']
X_train, X_test, y_train, y_test=train_test_split(X, y, train_size=0.8, random_state=69, stratify=y)
```

```
rnd_clf = RandomForestClassifier(n_estimators=200, max_leaf_nodes=128, min_impurity_decrease = 0.1, criterion = 'gini')
rnd_clf.fit(X_train, y_train)
y_pred = rnd_clf.predict(X_test)
```

```
# Predicted Data Value Counts
pd.Series(y_pred).value_counts()
```

```
City         20
Class4       10
Class2       10
Waterbody    10
```

```
Forest        8
Class3        8
Class5        8
Class1        8
dtype: int64
```

```
y_test.value_counts()
```

```
Class4       10
Class2       10
Waterbody    10
Grassland    10
City         10
Forest        8
Class3        8
Class5        8
Class1        8
Name: Rice-Groups, dtype: int64
```

```
# Accuracy Of The Model
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix, roc_auc_score, roc_curve, classification_report

accuracy_score(y_test, y_pred)
```

```
0.9878048780487805
```

```
f1_score(y_test,y_pred, average=None)
```

```
array([0.66666667, 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 0.        , 1.        ])
```

```
a = y_test
b = pd.get_dummies(a)
b = b.values.argmax(1)
```

```
b
```

```
array([4, 4, 5, 4, 3, 1, 2, 8, 0, 6, 2, 8, 5, 1, 2, 0, 2, 1, 8, 1, 4, 2,
       7, 1, 5, 7, 3, 4, 1, 7, 3, 8, 2, 6, 8, 5, 4, 8, 4, 1, 5, 1, 0, 1,
       6, 4, 3, 6, 2, 3, 3, 0, 8, 5, 0, 2, 4, 0, 0, 3, 1, 1, 1, 2, 1, 5,
       8, 4, 2, 6, 3, 0, 6, 5, 6, 4, 8, 1, 0, 1, 6, 5])
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
c = le.fit_transform(y_test)
```

```
c
```

```
array([4, 4, 5, 4, 3, 1, 2, 8, 0, 6, 2, 8, 5, 1, 2, 0, 2, 1, 8, 1, 4, 2,
       7, 1, 5, 7, 3, 4, 1, 7, 3, 8, 2, 6, 8, 5, 4, 8, 4, 1, 5, 1, 0, 1,
       6, 4, 3, 6, 2, 3, 3, 0, 8, 5, 0, 2, 4, 0, 0, 3, 1, 1, 1, 2, 1, 5,
       8, 4, 2, 6, 3, 0, 6, 5, 6, 4, 8, 1, 0, 1, 6, 5])
```

Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

```
parameters = {
    'criterion':('gini','log_loss','entropy'),
    'n_estimators':[200,300,400,500],
    'max_leaf_nodes':[64,128,256,512],
    'min_impurity_decrease':[0.1,0.2,0.3,0.01],
}
```

```
clf_gCV = GridSearchCV(rnd_clf,param_grid=parameters)
```

```
clf_gCV.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
## Prediction
y_pred=clf_gCV.predict(X_test)
print(confusion_matrix(y_pred,y_test))
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
[[ 9  0  0  0  0  0  0  0  0]
 [ 0  8  0  0  0  0  0  0  0]
 [ 0  0 10  0  0  0  0  0  0]
 [ 0  0  0  8  0  0  0  0  0]
 [ 0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  8  0  0  0]
 [ 0  0  0  0  0  0  8  0  0]
 [ 1  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0 10]]
0.9878048780487805
              precision    recall  f1-score   support

        City       0.90      1.00      0.95         9
      Class1       1.00      1.00      1.00         8
      Class2       1.00      1.00      1.00        10
```

```
      Class3       1.00      1.00      1.00         8
      Class4       1.00      1.00      1.00        10
      Class5       1.00      1.00      1.00         8
      Forest       1.00      1.00      1.00         8
   Grassland       1.00      0.91      0.95        11
   Waterbody       1.00      1.00      1.00        10

    accuracy                           0.99        82
   macro avg       0.99      0.99      0.99        82
weighted avg       0.99      0.99      0.99        82
```

```
clf_gCV.best_params_
```

```
{'criterion': 'log_loss',
 'max_leaf_nodes': 64,
 'min_impurity_decrease': 0.01,
 'n_estimators': 500}
```

NDVI

Choosing Validation Area

[ ] ↳ *4 cells hidden*

Data taking

[ ] ↳ *2 cells hidden*

Dataset preparation

[ ] ↳ *7 cells hidden*

Sentinel 2

↳ *5 cells hidden*