National Training Program on Advancing Paddy Mapping using open-source Earth Observation data and Geospatial Technologies

10 October 2025

Yellow River Auditorium, IWMI Colombo

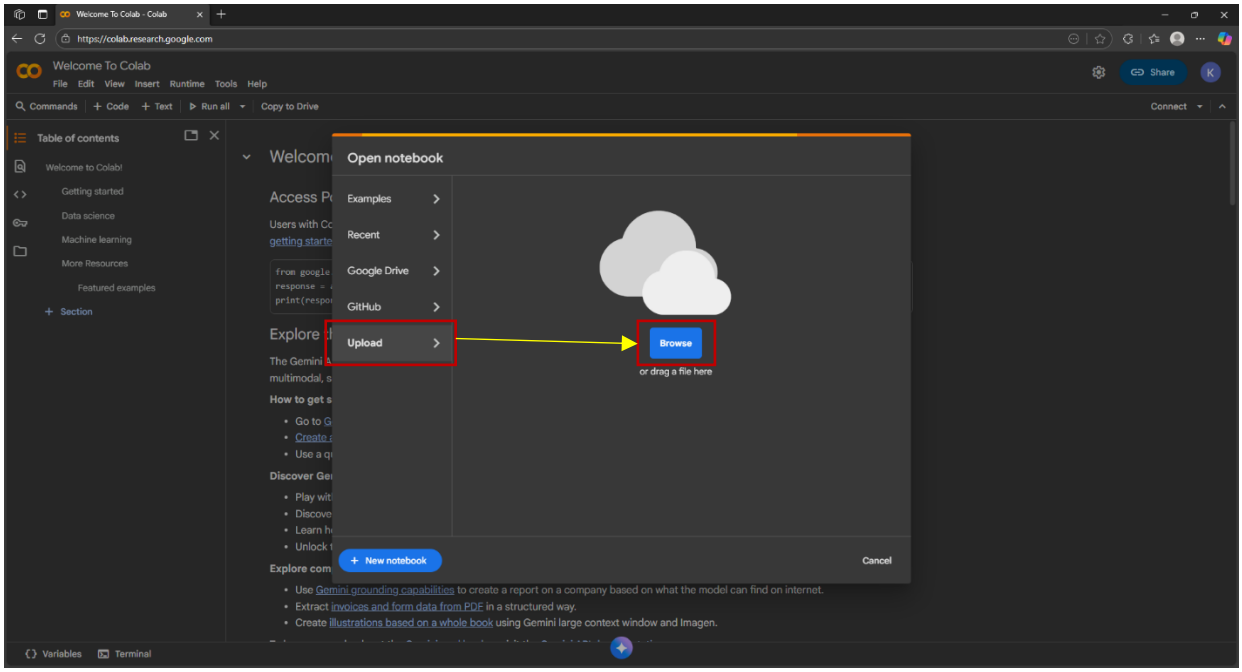
Training Guide

Table of Contents

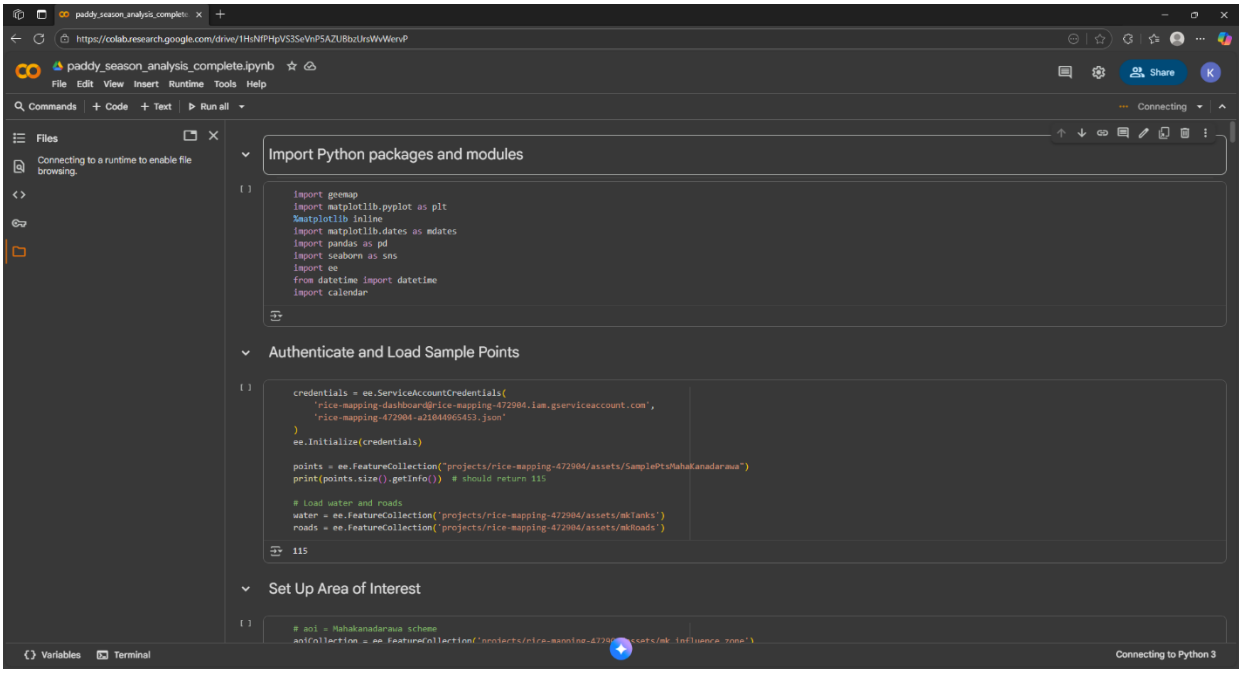
Run the GEE Notebook in Google Colab

1.1.Open the Notebook

- Go to Google Colab.
- Click “Upload → Browse”.

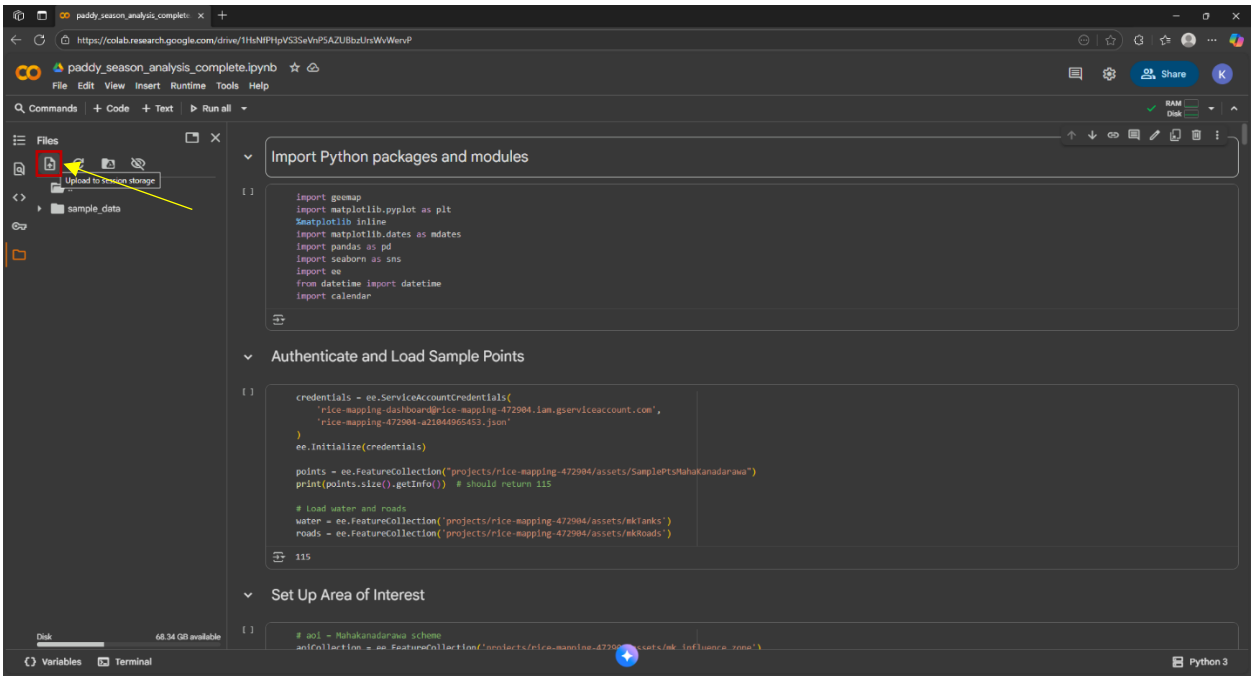


- Browse for the .ipynb file provided → Open.
- The notebook will open in Colab, ready to run.

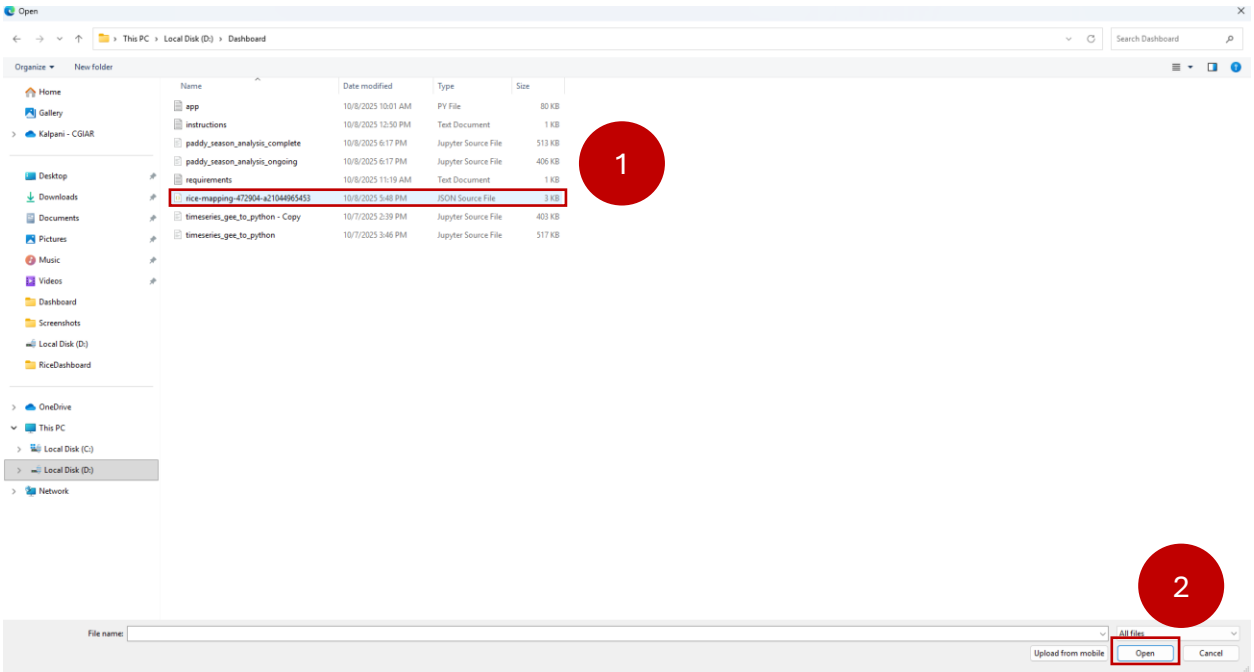


1.2.Upload the Service Account JSON Key

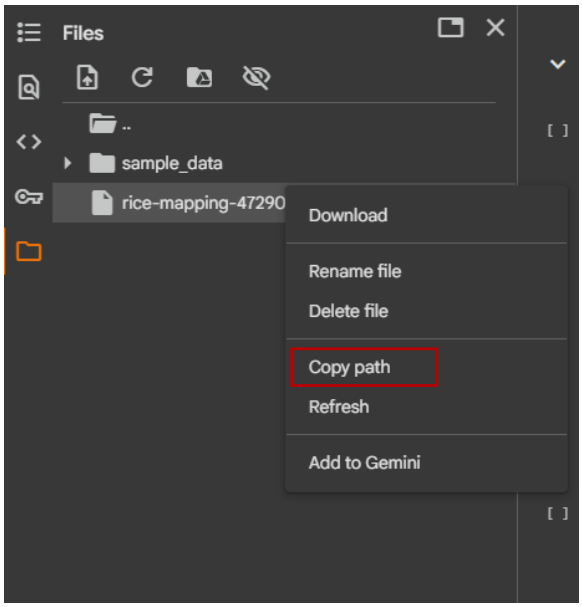
- In the Colab notebook, click “Upload to session storage”



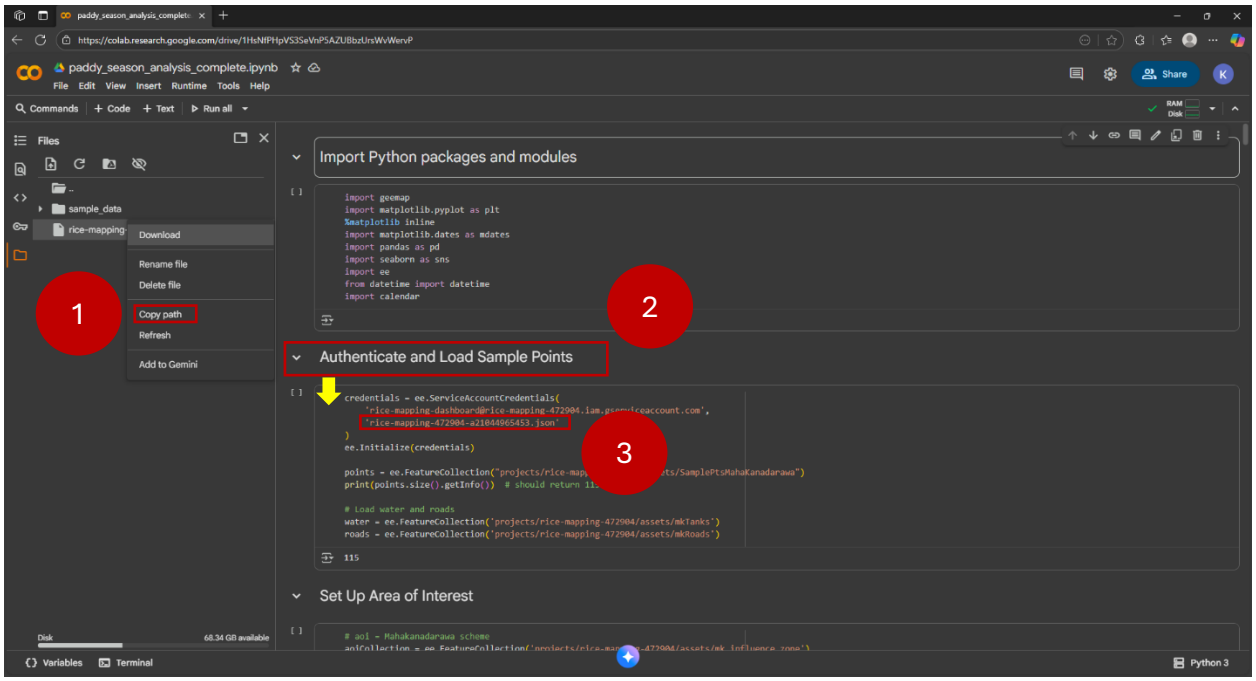
- Select the JSON key file → Open



- Right-click on the uploaded JSON file and select “Copy path.”



- Go to the cell in the notebook where it asks for the service account JSON key path.
- Paste this path into the notebook cell where it asks for the JSON key location.



1.3.Run the Notebook

- Start running the notebook cell by cell:
- Click on a cell.
- Press Shift + Enter to execute.
- Follow the instructions and observe outputs (maps, charts, or tables) in each cell.

Note: Make sure to run initialization/authentication cells first before any analysis cells, so the notebook can connect to Google Earth Engine properly.

1.4.Save Your Work

- You can save your changes by clicking File → Save a copy in Drive.

Setting Up Google Earth Engine Cloud Project and Authentication in Colab

2.1 Create a Google Earth Engine Cloud Project

You must first create a Cloud Project to connect the analysis and assets to Google Earth Engine (GEE).

Method 1:

- Go to Google Cloud Console.
- Enter a Project Name and click Create.
- Once created, note the Project ID. This will be used in your code.

Method 2:

- Open your browser and go to https://console.cloud.google.com/.
- Sign in using the Google account that you also use for Google Earth Engine (GEE).
- At the top of the Cloud Console page, click on the *navigation menu* (☰) → *IAM & Admin* → *Create a project*.
- In the popup window, click the "*New Project*" button.
- Fill in the following fields:
  - Project name: Enter something meaningful.
  - Location / Folder: If you don't have one, it may show *"No organization"*. That's fine. You can leave it as default.
  - Click *Create*.

2.1.1. Enable the Earth Engine API

- After creating the project, navigate to the *API & Services* → *Library* tab in Cloud Console.
- Search for "Earth Engine API."
- Click Enable. This step allows Colab or Python scripts to access Earth Engine resources using your project.

2.1.2. Link the Project with Earth Engine

- Go to the Earth Engine Code Editor.

- Click your profile icon → *Change Cloud Project*.
- Add the new project. You can now upload assets (FeatureCollections, Images, etc.) to that project.

2.1.3. Upload Assets to the Project

- In the Earth Engine Code Editor, go to the Assets tab.
- Click "NEW" → "Shape/CSV/GeoTIFF Upload."
- Select your local file and assign it to your project folder.
- Click Upload, and wait for processing to complete.

2.1.4. Authenticate and Initialize Earth Engine in Google Colab

- Once your project and assets are ready, open a Colab notebook and run:
- `import ee`
- `ee.Authenticate()`: will prompt you to log in to your Google account
- `ee.Initialize(project=PROJECT-ID')` : connects Colab to your Earth Engine Cloud Project

2.1.5. Load and Verify Your Assets

Once authenticated, you can directly load assets from your project using their paths:

- `points = ee.FeatureCollection('/path ')`
- `water = ee.FeatureCollection('/path ')`
- `roads = ee.FeatureCollection('/path ')`
- `aoiCollection = ee.FeatureCollection('/path ')`

These lines will load your point samples, water bodies, roads, and area of interest (AOI) into Earth Engine for analysis.

Detailed Notebook Explanation

This code cell executes imports Python packages, libraries and modules used in the analysis.

- Module: a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

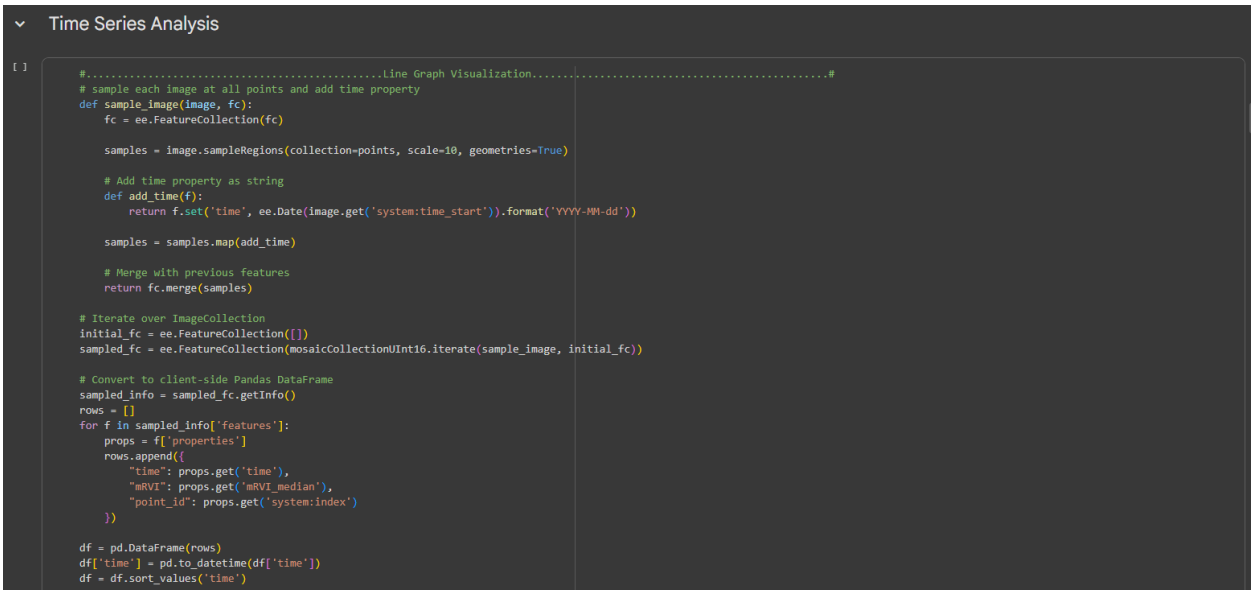
- Package: a way to organize and structure code by grouping related modules into directories. A package is essentially a folder that contains an __init__.py file and one or more Python files (modules). It allows modules to be easily shared and distributed across different applications.
- Library: a collection of codes or modules of codes that we can use in a program for specific operations.
- Modules → calendar, datetime,ee, Packages → matplotlib, seaborn, Libraries → geemap, pandas

Notes on libraries: `geemap` / `ee`- Google Earth Engine and geemap for accessing satellite data, visualization, and server-side computations. `pandas`- used for tabular data manipulation, DataFrame creation, and time-series handling. `matplotlib` - plotting time series, histograms, and saving figures.

This code cell imports the Google Earth Engine Python API so you can access GEE functions in your Python environment. It will open a browser window to sign in with your Google account and grant access to your GEE assets. This step is required once per session (or per machine) to verify your identity. 'ee.Initialize(project='PROJECT ID')' Initializes the GEE API in Python. It connects your session to a specific GEE project (replace 'PROJECT ID' with your Google Cloud project ID). After this, you can access datasets, images, and assets from your GEE account.

This code defines the Area of Interest (AOI), here labeled as *Mahakanadarawa scheme*, using a shapefile.

“Set Start and End Dates” explains that the analysis period is being defined. The next cell generates a list of *dekads* (12-day intervals per month) within the selected date range for temporal analysis and calculates mRVI.



“Time Series Analysis” marks the section focused on analyzing temporal patterns (e.g., mRVI over time). It also plots time-series line graphs, visualizing mRVI (or other indices) over time for selected points.

Get parameters

```
# Define dates once
dates = {
    'start': '2021-12-13',
    'peak': '2022-02-25',
    'harvest': '2022-04-01'
}
```

```
# Python datetime for pandas
start_date = pd.to_datetime(dates['start'])
peak_date = pd.to_datetime(dates['peak'])
harvest_date = pd.to_datetime(dates['harvest'])

# Convert time to datetime
df_long['time'] = pd.to_datetime(df_long['time'])

# ---------------------- Quantile Calculation ---------------------- #
start_values = df_long[df_long['time'] == start_date]['value']
peak_values = df_long[df_long['time'] == peak_date]['value']
harvest_values = df_long[df_long['time'] == end_date]['value']

# Calculate quartiles
q3_start = start_values.quantile(0.75)
q1_peak = peak_values.quantile(0.25)

print(f"Q3 of start date ({start_date.date()}): {q3_start}")
print(f"Q1 of peak date ({peak_date.date()}): {q1_peak}")

# ---------------------- Difference Calculation ---------------------- #
mean_start = start_values.mean()
mean_peak = peak_values.mean()
mean_harvest = harvest_values.mean()
```

“Get parameters” define and extract analysis parameters that is used later in the analysis. This requires important analysis dates including SOS, peak, harvest.

Rice Mapping

```
# Earth Engine dates
sosDate = ee.Date(dates['start'])
peakDate = ee.Date(dates['peak'])
fallDate = ee.Date(dates['harvest'])

#.....................................................mRVI SOS-Peak-Fall analysis.....................................................#
#  Function to get adjacent dekads
def getAdjacentDekads(targetDate, dekadList):
    index = dekadList.indexOf(targetDate)
    return ee.List([
        dekadList.get(ee.Number(index).subtract(1)),
        targetDate,
        dekadList.get(ee.Number(index).add(1))
    ]).filter(ee.Filter.neq('item', None))

# Extract SOS, Peak, Fall Images
sosWindow = getAdjacentDekads(sosDate, filteredDekadList)
sosImages = mosaicCollectionUInt16.filter(ee.Filter.inList('dekad', sosWindow))
sosMin = sosImages.reduce(ee.Reducer.min())

peakWindow = getAdjacentDekads(peakDate, filteredDekadList)
peakImages = mosaicCollectionUInt16.filter(ee.Filter.inList('dekad', peakWindow))
peakMax = peakImages.reduce(ee.Reducer.max())

fallWindow = getAdjacentDekads(fallDate, filteredDekadList)
fallImages = mosaicCollectionUInt16.filter(ee.Filter.inList('dekad', fallWindow))
fallMin = fallImages.reduce(ee.Reducer.min())

# Main Conditions
positiveGrowth = peakMax.subtract(sosMin).gt(diff_start_peak/2)
negativeDecline = peakMax.subtract(fallMin).gt(diff_peak_harvest/2)
```

“Rice Mapping” section is related to mapping rice-growing areas and start of rice cropping. It uses previously defined dates and Earth Engine date functions to handle temporal layers for rice mapping.

Visualize maps

```
# Visualize mRVI Data
rviVis = {"min": 0.0, "max":8000, "palette": ['FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901', '66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01', '

m.add_basemap("Esri.WorldImagery")
m.addLayer(mosaicCollectionUInt16.median().clip(aoi), rviVis, 'mRVI', False)
# m.addLayer(paddyMask, {"min": 0, "max": 1, "palette": ['red', 'green']}, '(mRVI SOS-Peak-Fall) Paddy Map')
m.addLayer(maskedPaddyClassification, {"min": 0, "max": 1, "palette": ['green']}, 'Masked Paddy Map')
m.addLayer(waterMask.updateMask(waterMask), {"palette": ['#0000FF']}, 'Water Mask', False)
m.addLayer(roadsMask.updateMask(roadsMask), {"palette": ["#FDEE69"]}, 'Roads Mask', False)
# m.addLayer(maskedLongest, {"min": 0, "max": 10, "palette": ['white', 'blue']}, 'Longest Streak', False)
# m.addLayer(maskedStartDate, {"min": 1638297000000, "max": 1653935400000, "palette": ['#ffeda0', "#bd008e"]}, 'Longest Streak Start Date', False)
m.addLayer(maskedStartMonth, {"min": 1, "max": 12, "palette": ["blue", "cyan", "green", "lime", "yellow", "orange", "red", "pink", "purple", "brown", "gray", "black"]}, "Start Month", Fal
m.addLayer(maskedStartMonthDay, {"min": 101, "max": 1231, "palette": ["blue", "cyan", "green", "yellow", "orange", "red"]}, "Start MMDD", False)

m
```

This cell visualizes mRVI and rice-classified maps on an interactive map interface.

This section computes total paddy area by summing rice-classified pixels over the AOI.

Rice Mapping Dashboard

3.1. Launching the Dashboard

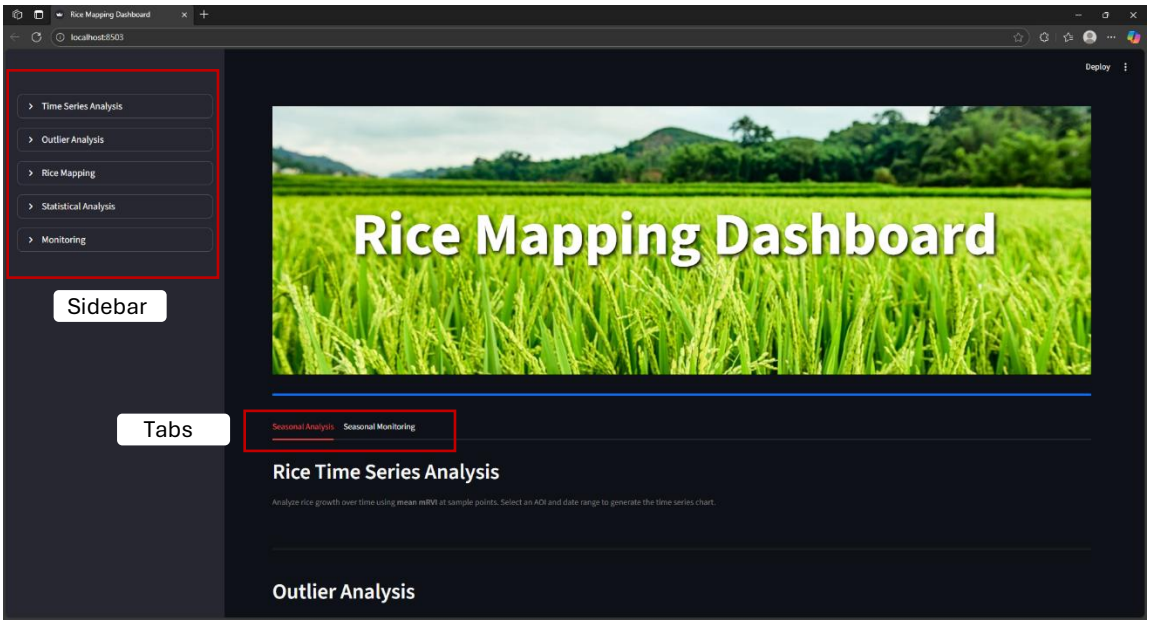
This chapter guides you through accessing the Rice Mapping Dashboard. The dashboard is built using **Python** and **Streamlit**, integrated with **Google Earth Engine (GEE)** for remote sensing data processing.

3.1.1. Open the Dashboard

- Use the link; Rice Mapping Dashboard · Streamlit to open the dashboard in your default browser. You will see the main interface with the sidebar and map.

3.1.2. Dashboard Overview

- A **sidebar** (for inputs and controls)
- Two main **tabs**: Seasonal Analysis and Seasonal Monitoring
- Visualizations, graphs, and interactive maps are displayed once you run the analysis.



Dashboard Overview

3.2. Exploring the Dashboard

3.2.1. An Overview of the Interface

The Rice Mapping Dashboard is designed to make the analysis of Sentinel-1 based rice growth patterns user-friendly. It provides tools to visualize time-series data, detect seasonal trends, and map rice extent.

The Sidebar

The sidebar is divided into five expandable sections:

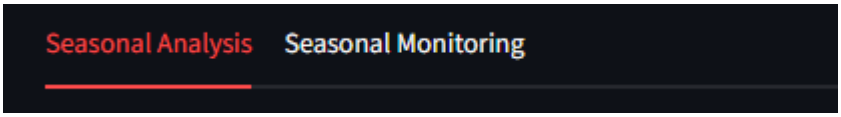
	Section	Description
1	Time Series Analysis	Set AOI (Area of Interest), start & end dates, and run the temporal analysis.
2	Outlier Analysis	Visualize the dispersion of mRVI values and identify outliers.
3	Rice Mapping	Select Start, Peak, and Harvest dates to generate an accurate paddy classification map for the season.
4	Statistical Analysis	Compute total paddy area and analyze cropping patterns by date/month.
5	Monitoring	Analyze in-season rice growth for ongoing seasons.

Sidebar View

The Tabs

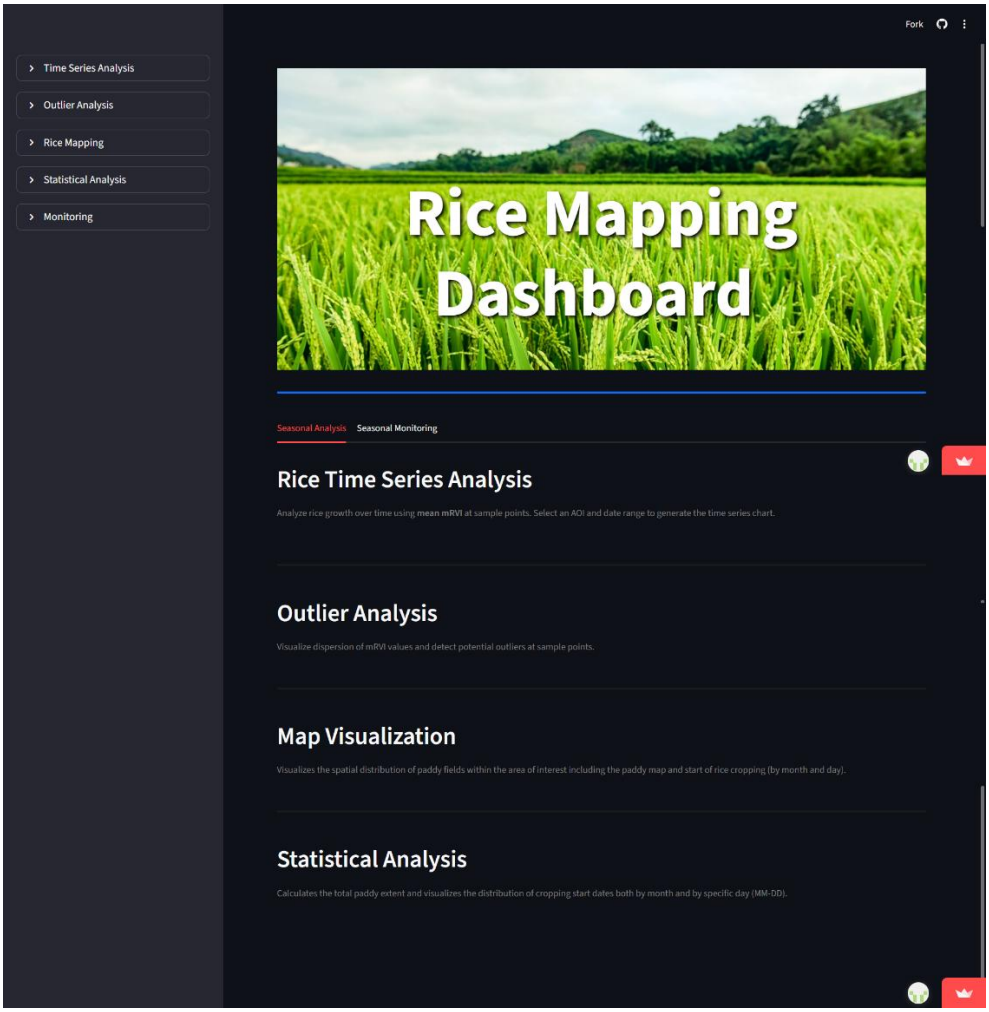
The dashboard has **two main tabs** that represent two modes of operation:

Tab	Purpose
Seasonal Analysis	For retrospective analysis of a completed rice season. Includes detailed time-series, outlier, and statistical modules.
Seasonal Monitoring	For real-time monitoring of an ongoing or recent season. Automatically detects the Start of Season (SOS) and peak periods.



Seasonal Analysis and Monitoring Tabs

3.2.2. Seasonal Analysis

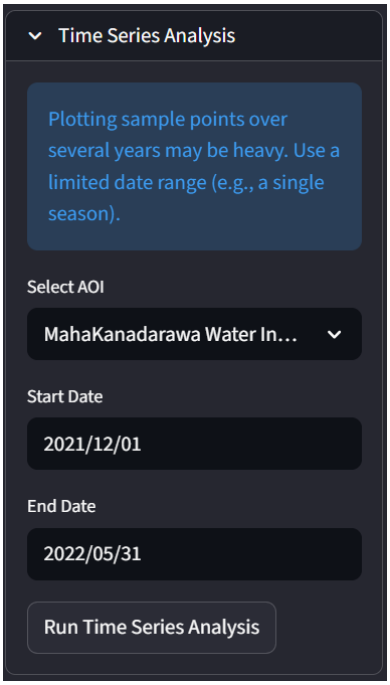


Performs retrospective analysis of a completed season, produce time series, flag outliers, classify paddy fields from Start/Peak/Harvest dates and compute area statistics.

Subsection	Description
Rice Time Series Analysis	Computes mean mRVI values across sample points over time. Visualizes temporal variations of rice growth.
Outlier Analysis	Generates boxplots to identify abnormal mRVI values (noise or non-rice areas).
Rice Mapping	Combines mRVI changes between Start, Peak, and Harvest dates to map paddy areas.
Map Visualization	Displays classified paddy areas, start-of-season month/day maps, and temporal growth layers on an interactive GEE map.
Statistical Analysis	Calculates total paddy area (in hectares) and provides charts of paddy area by month and date.

Time Series Analysis

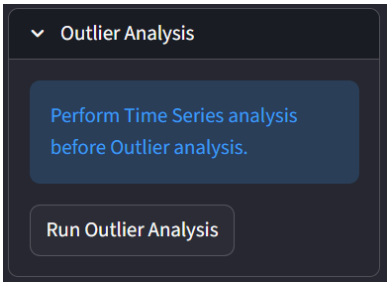
- Sidebar → *Time Series Analysis* → select AOI, choose Start Date and End Date (use a single season window) → click *Run Time Series Analysis*.



- The app builds dekadal mosaics from Sentinel-1, calculates (mRVI), samples the sample points, and draws two plots:
  - Line plot of mean mRVI over time.
  - Point time series for each sample point.

Outlier Analysis

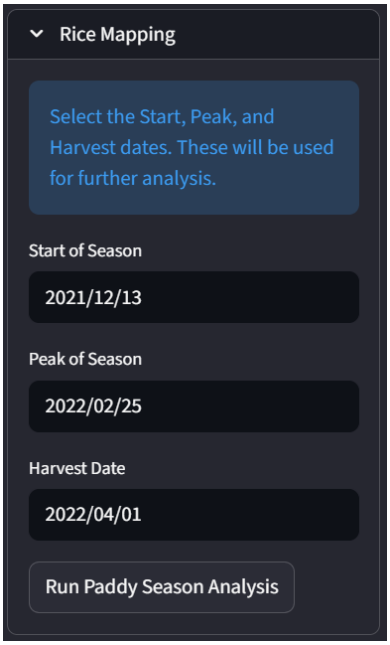
- Sidebar → *Outlier Analysis* → *Run Outlier Analysis*.



- The app draws mRVI dispersion as a boxplot to spot outlier points.

Rice Mapping

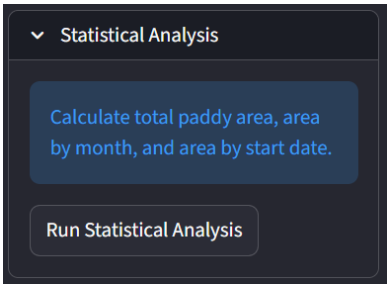
- Sidebar → *Rice Mapping* → choose Start of Season, Peak of Season, Harvest Date → click *Run Paddy Season Analysis*.



- The app reduces computes quantiles at the selected dates (Q3 of start, Q1 of peak) and means to build thresholds, creates logical masks, cleans the paddy map, dilates, removes small objects, and masks water bodies and roads.

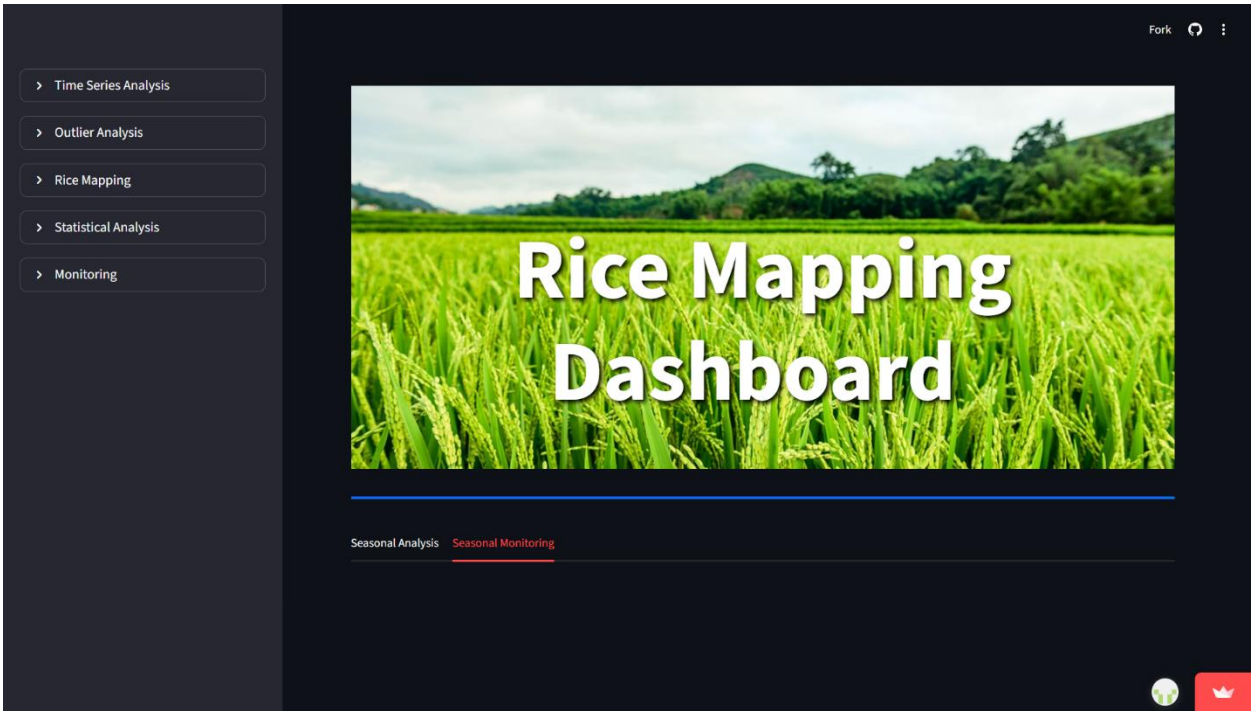
Statistical Analysis

- Sidebar → *Statistical Analysis* → *Run Statistical Analysis*.



- Total paddy extent, Area by Month, and Area by MM-DD are calculated.

3.2.3. Seasonal Monitoring

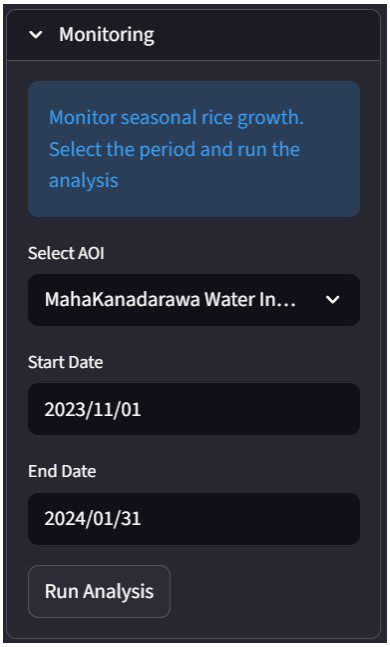


Perform near real-time monitoring for an active season, detect Start of the season automatically, monitor rise to peak, map current paddy extent and start dates.

Subsection	Description
Time Series Analysis	Plots mRVI variation for current-season dekads to identify growth trends.
Outlier Analysis	Detects anomalies in mRVI values during monitoring.
Parameter Extraction	Automatically detects Start of Season (SOS) and Peak date based on mRVI minima and maxima.
Map Visualization	Maps active paddy fields and seasonal growth onset using mRVI-derived patterns.
Statistical Analysis	Calculates current-season paddy area and its temporal distribution.

Monitoring

- Sidebar → *Monitoring* → Select AOI, Start Date, End Date → *Run Analysis*



- Set AOI and Date Range: In the sidebar, open Monitoring, select the AOI, choose the start and end dates of the monitoring period, and click *Run Analysis*.
- Automatic SOS Detection: The app creates dekadal mRVI mosaics and plots the median mRVI time series. It automatically detects the Start of Season (SOS) as the first local minimum and assigns the last date as the peak.
- Parameter Extraction: The app calculates quartiles (Q3 at SOS, Q1 at Peak) and mean differences to form thresholds, then builds logical masks for positive growth, negative decline, and value pattern.
- Map Visualization: Displays an interactive map with layers.
- Area and Statistics: Computes the total paddy area (ha) and the area distribution by month and date.