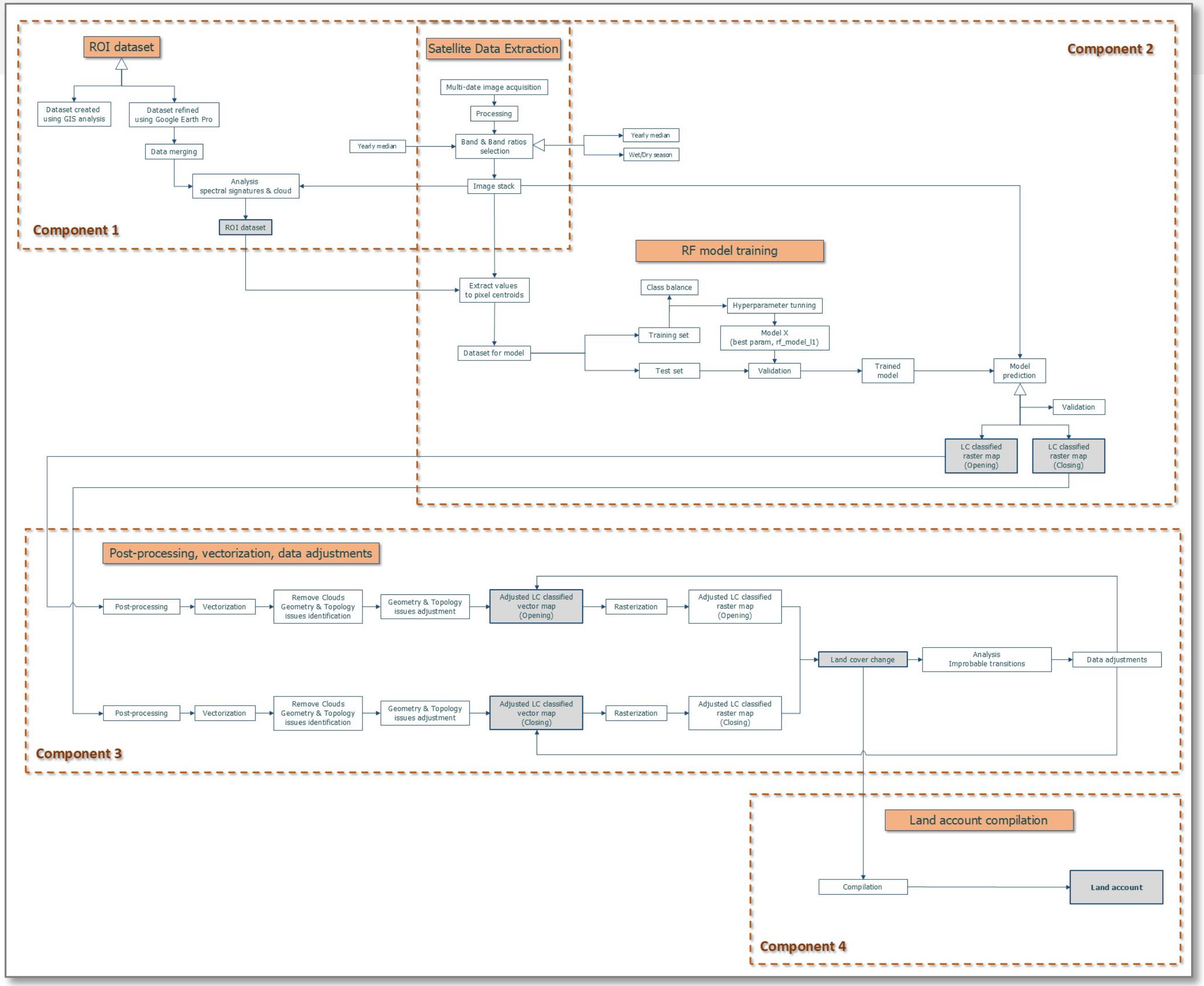
Land cover and land accounting in Vanuatu – Day 2

Components 1 (cont.) & 2

Blanca Perez-Lapena, PhD

April 1, 2025

Pipeline for Agile Estimation of Land Accounts (PAELA)



	Dense_Forest	Open_Forest	Forest_plantations	Mangroves	Agriculture	Coconut_Plantation	Grassland	Built-up_Infrastr	Water_body	Shrubs	Bareland	Total
Opening area	274316.9	13137.4	10666.8	752.5	375.6	60.0	1453.4	37753.4	16096.2	9720.2	20416.3	387577.9
Expansions	11301.9	24893.4	5267.3	652.0	173.9	69.1	865.7	12010.2	11082.0	10793.9	7446.6	86209.8
Regressions	18946.6	3582.7	4458.4	430.2	284.3	31.6	663.1	33856.4	9637.2	3494.5	9476.7	86209.8
Net change	7644.7	-21310.7	-808.9	-221.8	110.4	-37.5	-202.6	21846.1	-1444.8	-7299.3	1724.4	0.0
Closing area	266672.3	34448.1	11475.7	974.4	265.2	97.5	1656.1	15907.3	17541.0	17019.6	18691.9	387577.9

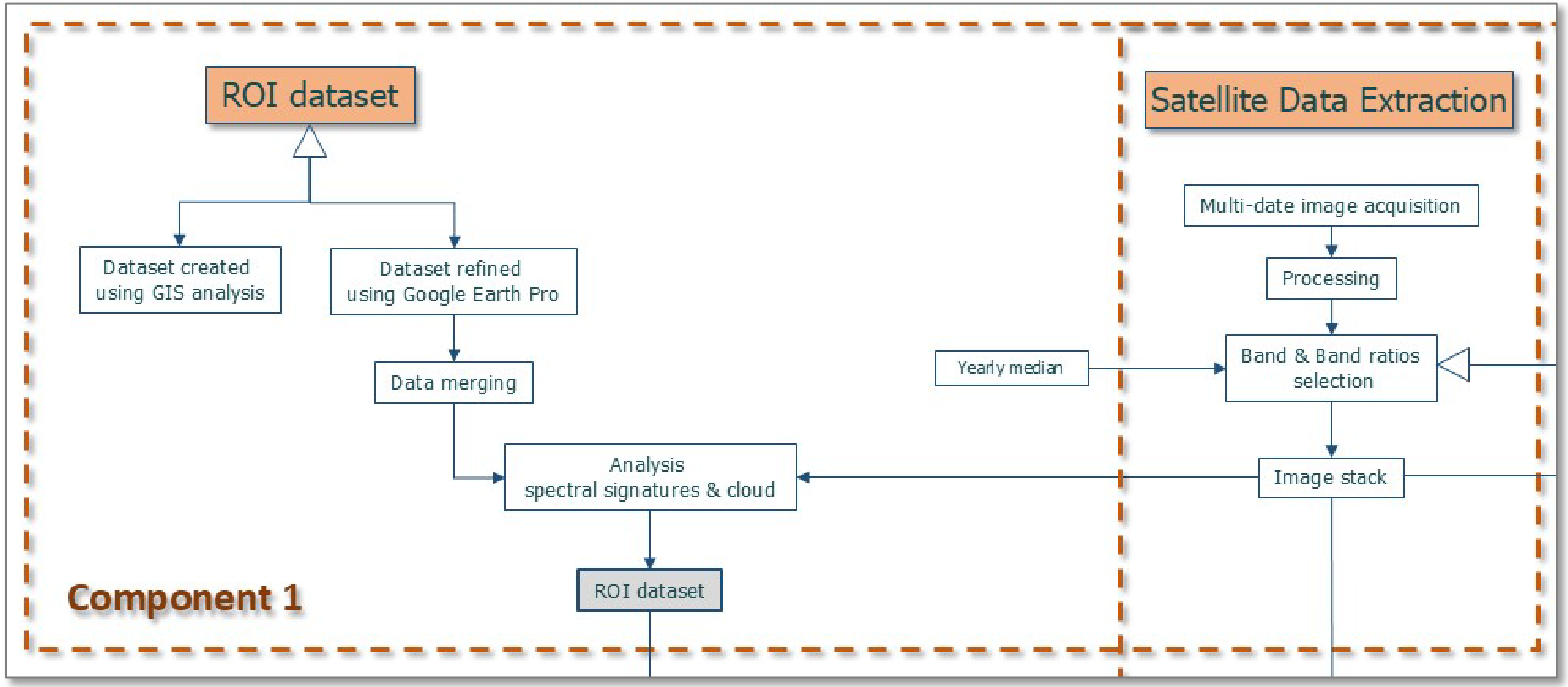
Yesterday – Component 1 (2020)

Step 1.1

Python

Step 1.2

PyQGIS

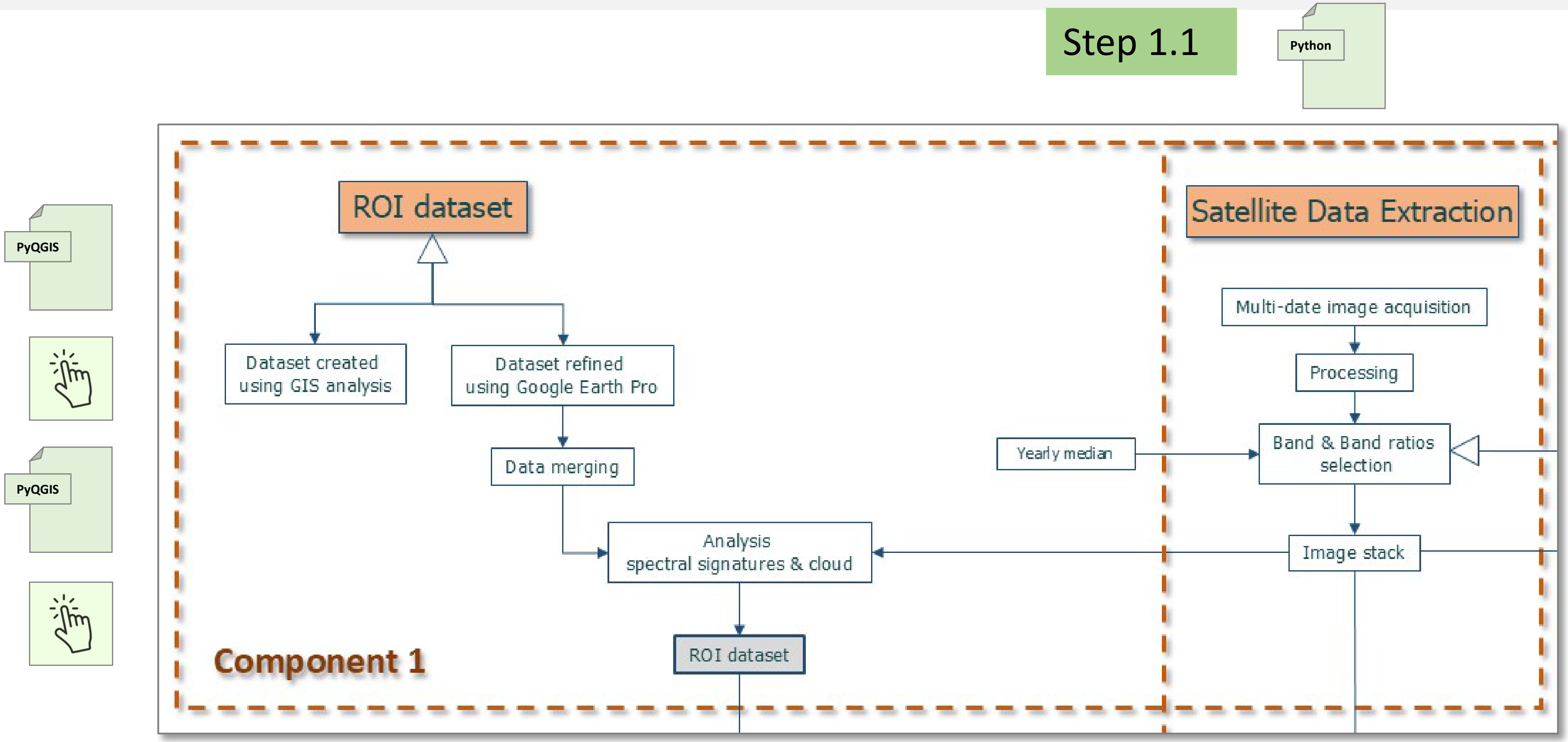


Today – Component 1 cont. (2020)

Step 1.1

Step 1.2

Step 1.3



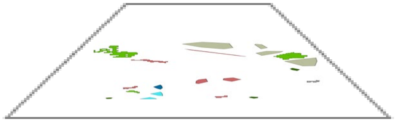
Today – Component 1: From vROIs2 (2020) to vROIs3 (2020)

Step 1.2

Step 1.3

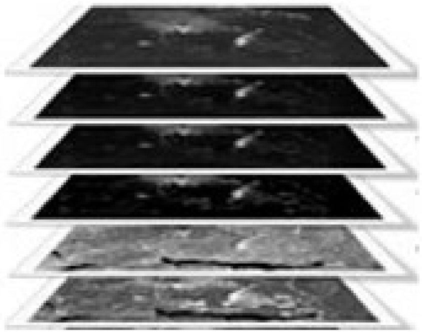
ROIs dataset

With 2020
cloud ROIs



Step 1.1 2020

Imagery

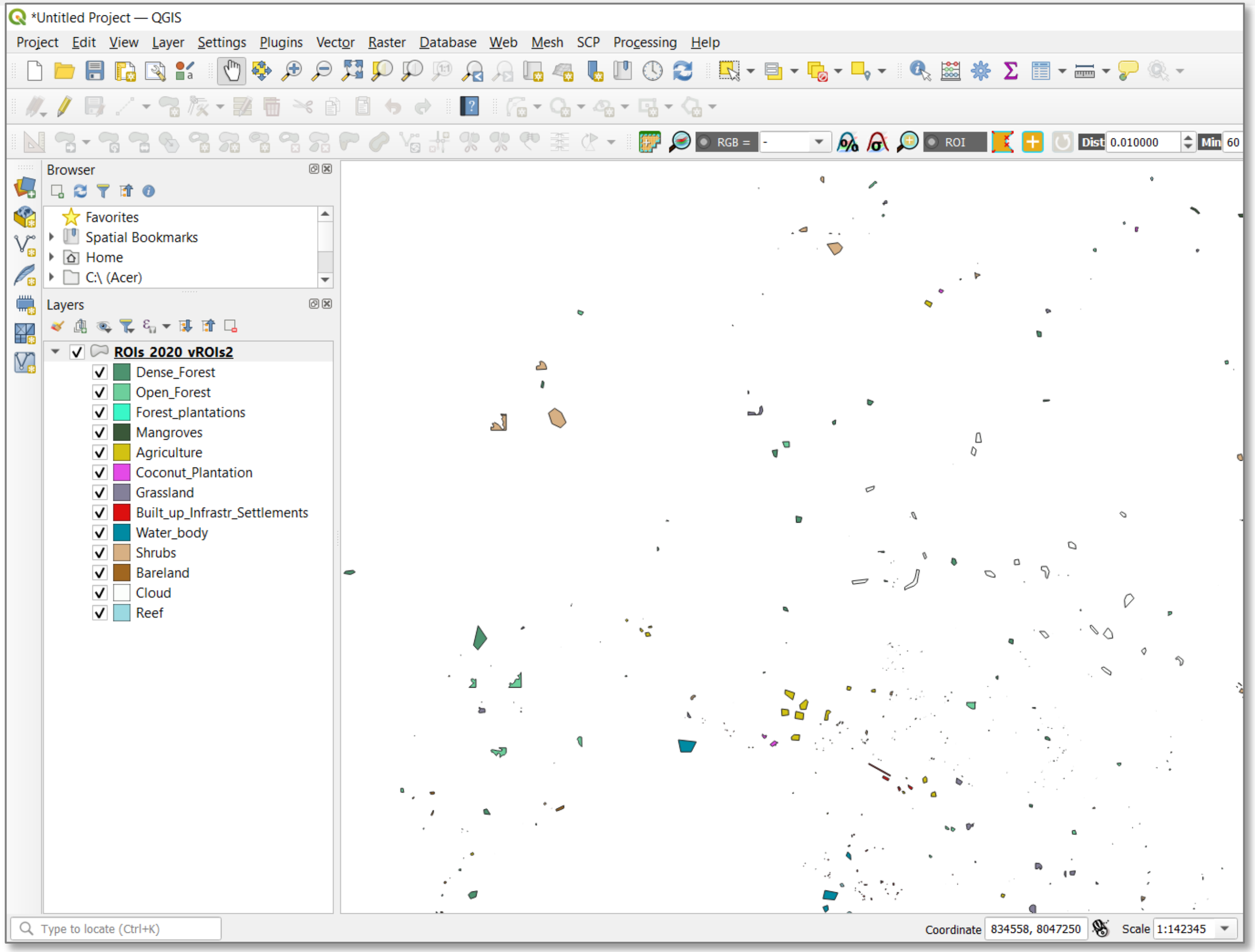


Yearly

Median Single bands

- Red
- Green
- Blue
- NIR
- SWIR

Today – Component 1: Output Refined ROIs (2020)

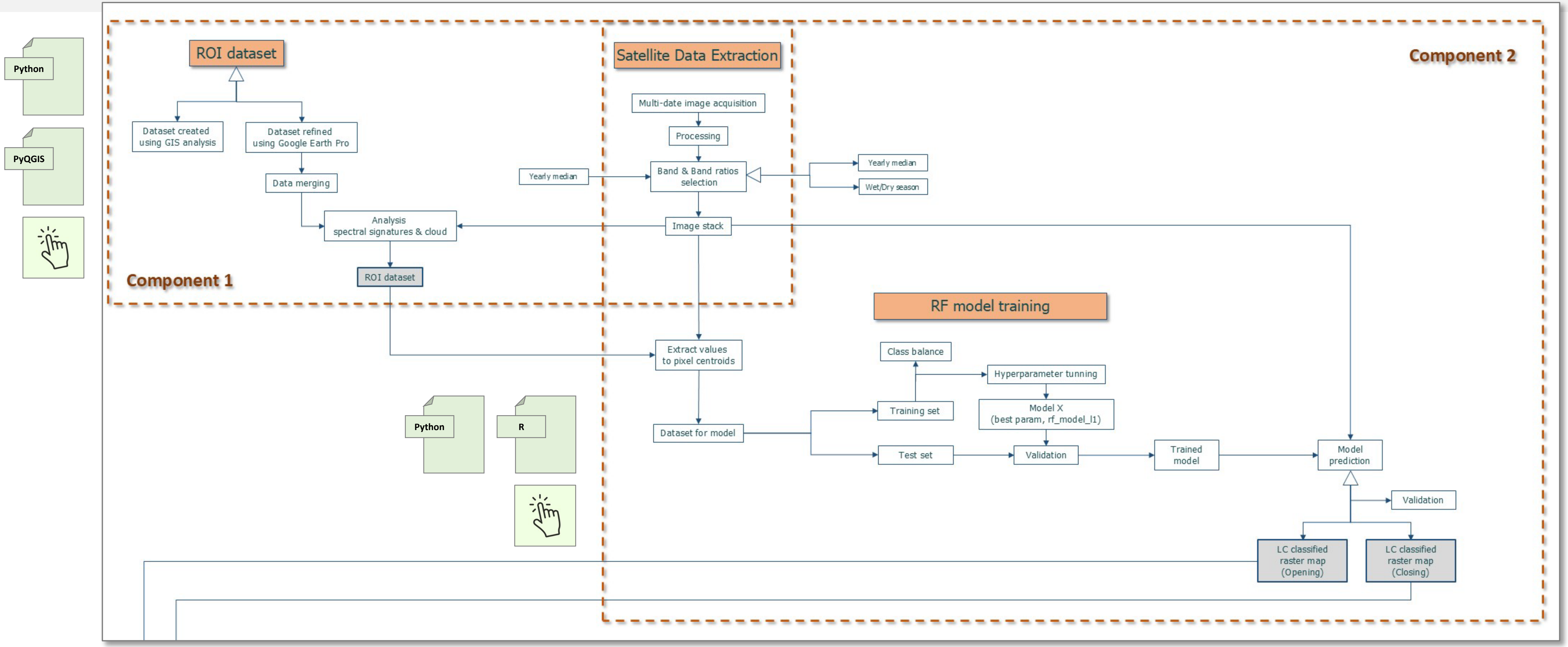


Today – Your turn

1) Component 1: Create the refined ROIs dataset for 2020

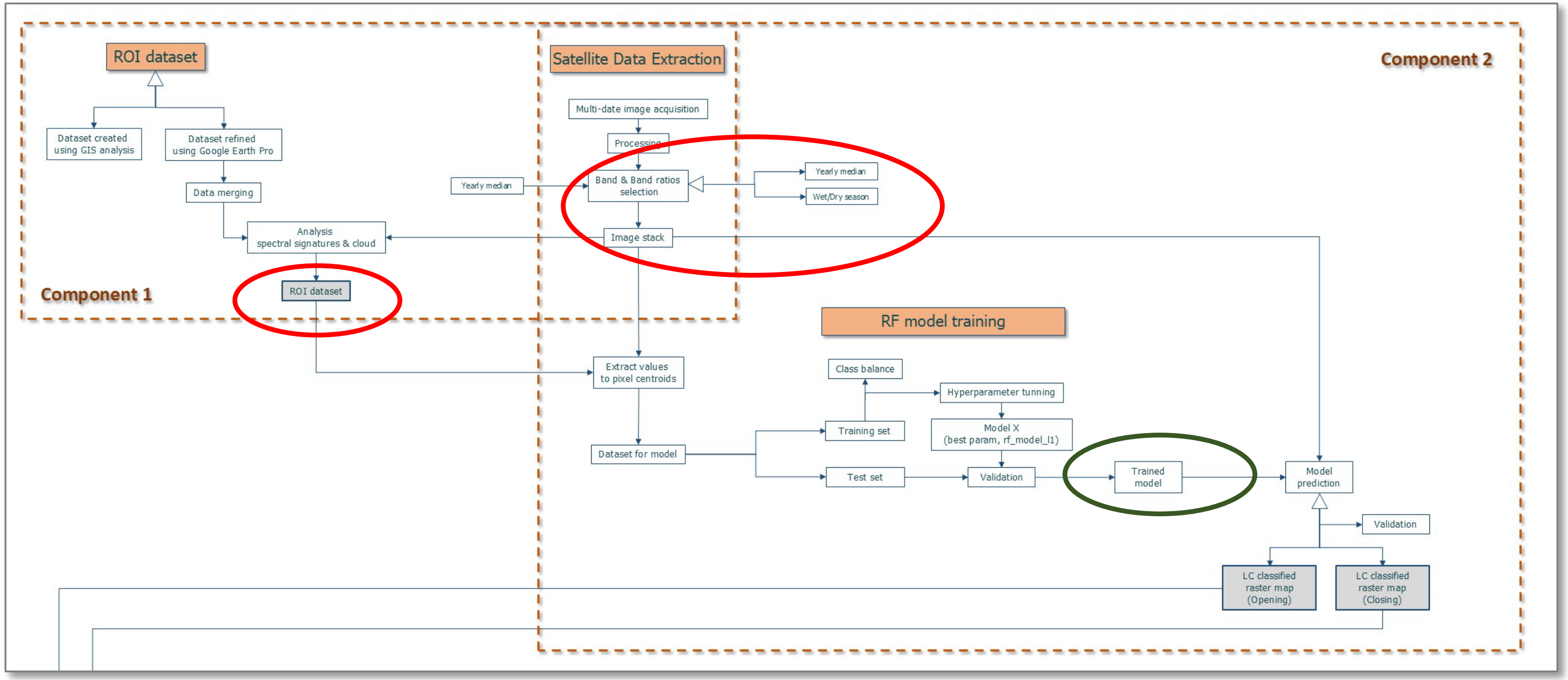
- Continue with Step 1.2:
  - FROM 'Z_Visit_Vanuatu_April2025\Component_1\output\2020\vROIs2'
  - TO 'D:\Z_Visit_Vanuatu_April2025\Component_1\output\2020\vROIs3'
  - Edit the kml in Google Earth Pro
- Go through Step 1.3

This training



Component 2: Satellite data extraction & RF model training & prediction

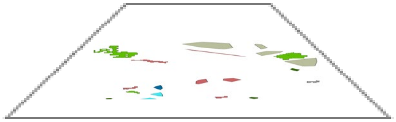
- Input:
  - ROIs dataset
  - Imagery stack



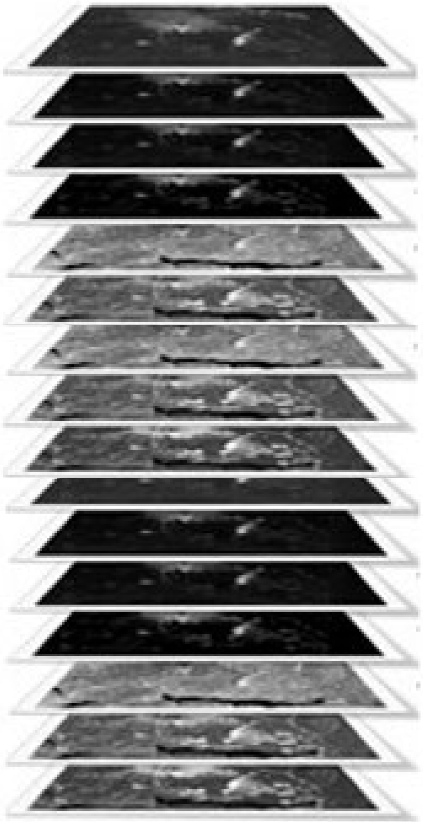
- Output:
  - Trained Random Forest model (e.g., Opening year)

Component 2: Input

ROIs dataset



Imagery



Yearly

Median Single bands

- Red
- Green
- Blue
- NIR
- SWIR

Median Indices

- NDVI (Normalized Difference Vegetation Index)
- MNDWI (Modified/Normalize Difference Water Index)
- SAVI (Soil Adjusted Vegetation Index)
- BSI (Bare Soil Index)

Temporal indices:

Median VCP

- VCP (Vegetation Cover Proportion)

Median Dry season

Median Wet season

- NDVI
- MNDWI
- BSI

Component 2: Input

Parameters

```
# Define the year as a variable
the_year = 2020
# Define desired resolution (e.g., 20 meters)
target_resolution = 20

# Define parameters
collections = "COPERNICUS/S2_SR_HARMONIZED"
bbox_small = ee.Geometry.BBox(168.12, -17.85, 168.6, -17.4)
```

1. Stack for model training

Yearly Median

```
datetime_range_selectedmonths = ee.DateRange(f"{the_year}-01-01", f"{the_year}-12-31")

# Load Sentinel-2 collection in GEE
collection_raw = (
    ee.ImageCollection(collections)
    .filterBounds(bbox_small)
    .filterDate(datetime_range_selectedmonths)
    .filter(ee.Filter.lt("CLOUDY_PIXEL_PERCENTAGE", 100))  # No cloud filter here
)

# Instead of trying to extract the EPSG code directly, use the entire crs string
common_projection = ee.Projection(collection_raw.first().select(0).projection().crs())


#Set desired resolution and resampling method
collection = collection_raw.map(
    lambda image: image.resample('bilinear').reproject(
        crs=common_projection,
        scale=target_resolution
    )
)
```

NDVI Wet season

```
ee.Authenticate(auth_mode='notebook')
ee.Initialize()

# Define NDVI calculation function
def doNDVI(image):
    # Calculate NDVI using bands B4 (Red) and B8 (NIR)
    ndvi = image.normalizedDifference(["B8", "B4"]).rename("NDVI")
    return ndvi

# Apply NDVI calculation for each image in the collection
ndvi_collection = collection_scaled_wet.map(doNDVI)

# Calculate the median NDVI across all dates
median_ndvi = ndvi_collection.reduce(ee.Reducer.median()).rename("median_NDVI")

# Check the projection of the resulting NDVI composite
ndvi_projection = median_ndvi.projection().getInfo()
print("NDVI Projection:", ndvi_projection)

## To reproject it to the original CRS:
#To keep the original (e.g., 10m)
#median_ndvi = median_ndvi.reproject(crs=original_crs, scale=projection["nominalScale"])
#To set it to the same as the other bands in the stack (e.g., 20m)
median_ndvi_reprojected_wet = median_ndvi.reproject(crs=original_crs, scale=target_resolution)
projection = median_ndvi_reprojected_wet.projection().getInfo()
print("NDVI Wet Projection information:", projection)
ndvi_back2original_crs = projection["crs"]
print("NDVI Wet Projection:", ndvi_back2original_crs)
```

Component 2: Model training -> Output trained model

```python
# --------------------- MODEL TRAINING ---------------------
print("Before dropping:")
print(data.columns)

columns_to_remove = ['class', 'x', 'y', 'geometry', 'crs']
data_features = data.drop(columns=columns_to_remove)

print("\nAfter dropping:")
print(data_features.columns)

X = data_features.values
y = data['class'].values # Labels (class IDs)

# Inspect the data
print("Features (X):", X.shape)
print("Labels (y):", y.shape)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100],  # Number of trees in the forest
    'criterion': ['gini'],  # Function to measure the quality of a split
    'max_depth': [13],  # Maximum depth of the tree
    'min_samples_split': [2],  # Minimum samples required to split an internal node
    'min_samples_leaf': [1],  # Minimum samples at a leaf node
    'max_features': ['sqrt'],  # Features to consider when looking for the best split
    'bootstrap': [True],  # Whether to bootstrap samples
    'random_state': [42]  # Seed for reproducibility
}

# Perform Grid Search
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Unique labels in training set:", np.unique(y_train))
print("Unique labels in test set:", np.unique(y_test))

# Retrieve the best model
best_rf_classifier = grid_search.best_estimator_
```

Model evaluation

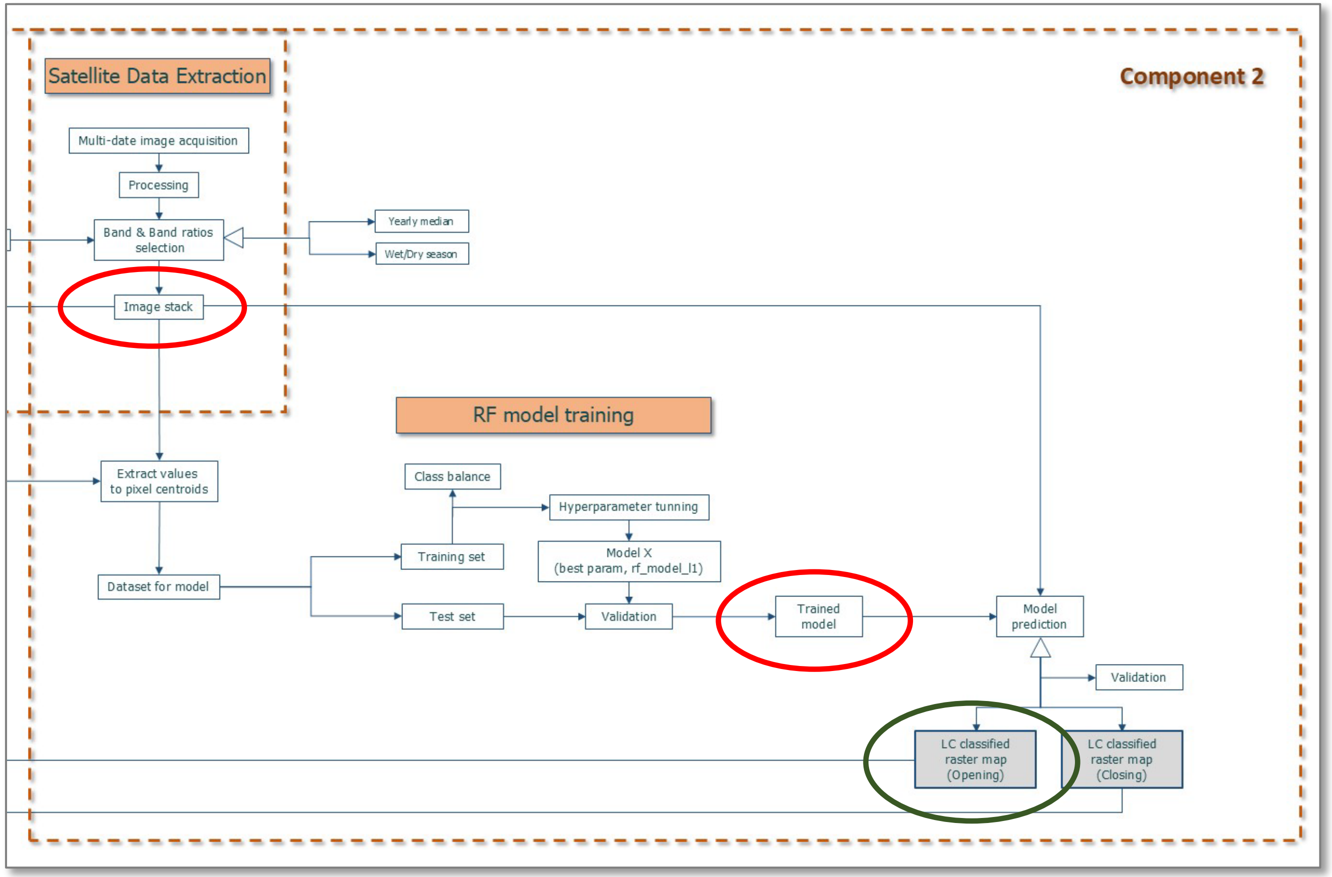
```python
# Model evaluation on the test set
y_pred = best_rf_classifier.predict(X_test)
print("Test Set Results:")
print(classification_report(y_test, y_pred))
# Get the classification report
report = classification_report(y_test, y_pred, output_dict=True)
# Print only the accuracy and confusion matrix
print("Accuracy:", report['accuracy'])
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```python
#Construct the file name dynamically
drive_folder = "/content/drive/My Drive/GEE_exports_COMPONENT_2"
model_filename = f"VANmodel_{the_year}_vROIs{version_num_ROIs}.joblib"
model_path = os.path.join(drive_folder, model_filename)
```

```python
# Save model to Colab first
dump(model_metadata, model_filename)
# Copy the model to Google Drive folder
import shutil
shutil.copy(model_filename, model_path)
```

Component 2: Satellite data extraction & RF model training & prediction

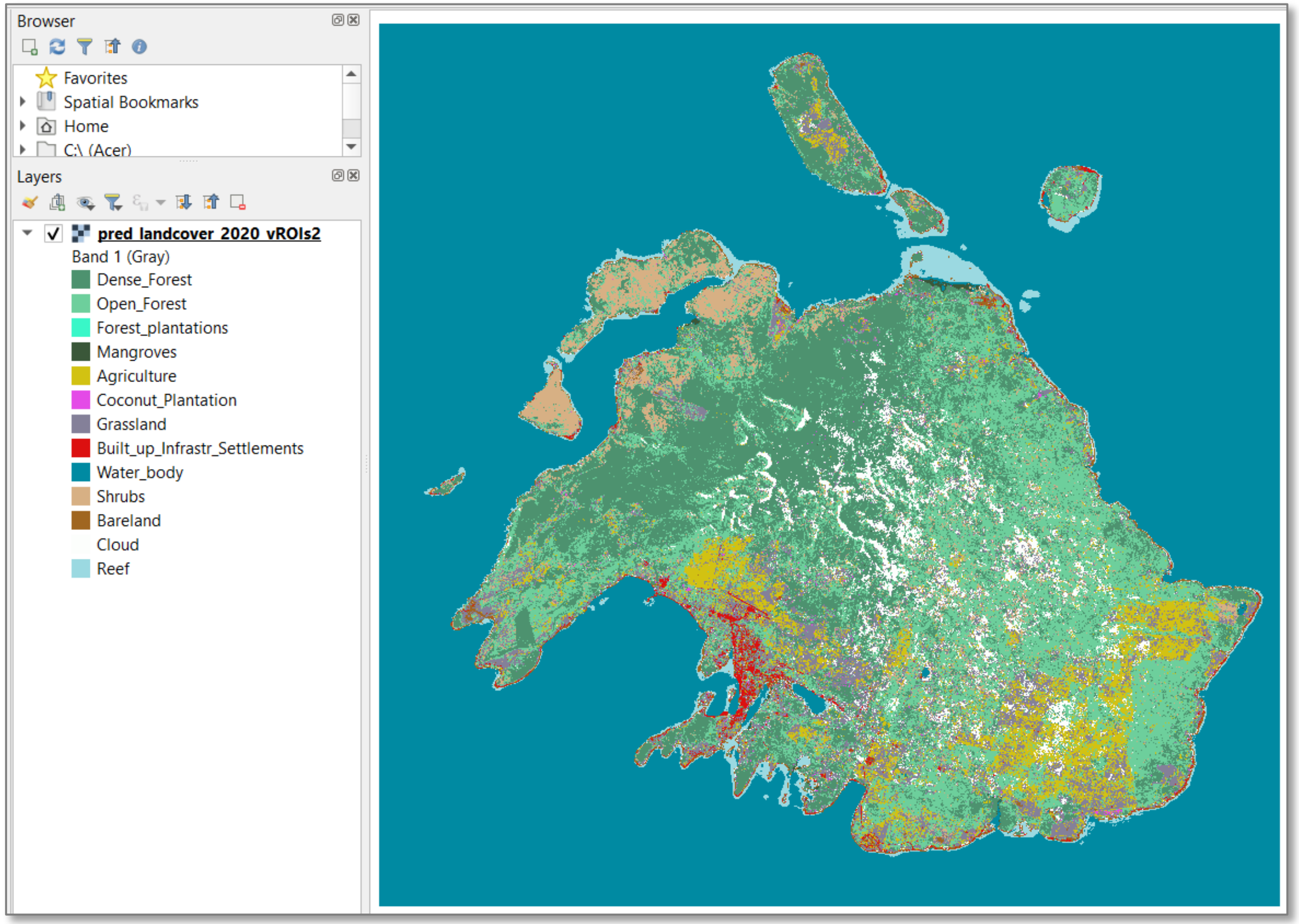
- Input:
  - Trained Random Forest model
  - Imagery stack (e.g., Opening year)
- Output:
  - Land cover classified raster map (e.g., Opening year)



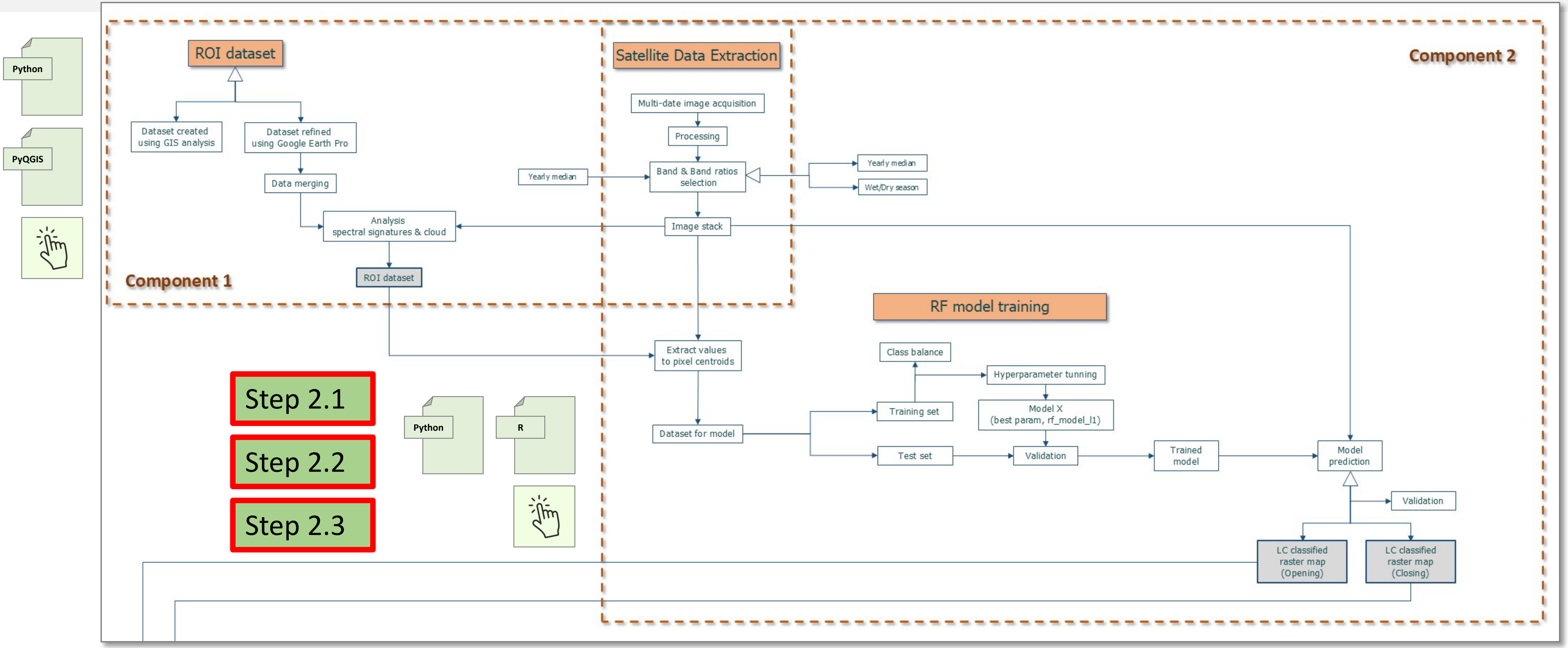
Component 2: Output Land cover classified raster map

Land cover classification

```
import rasterio
from rasterio.plot import show
import matplotlib.pyplot as plt
import numpy as np


# Load the raster stack for prediction
with rasterio.open(f"stacked_image_{the_year}_{target_resolution}m.tif") as src:
    raster_data = src.read()
    raster_profile = src.profile

# Reshape the raster data for prediction
raster_data = raster_data.reshape(raster_data.shape[0], -1).transpose()
# **Handle NaN values before prediction**
# Replace NaN with a specific value (e.g., 0)
raster_data = np.nan_to_num(raster_data)
# Make predictions using the loaded model
predictions = rf_model.predict(raster_data)

# Reshape predictions back to the original raster shape
predictions = predictions.reshape(src.height, src.width)

# Update raster profile for the output GeoTIFF
raster_profile.update({
    'dtype': rasterio.uint8,  # Assuming your labels are integers
    'count': 1,  # Single band for the classification
    'nodata': 0  # Set nodata value if needed
})

# Save predictions as a GeoTIFF
with rasterio.open(f"predicted_landcover_{the_year}_{target_resolution}m.tif".tif', 'w',
    dst.write(predictions.astype(rasterio.uint8), 1)
```



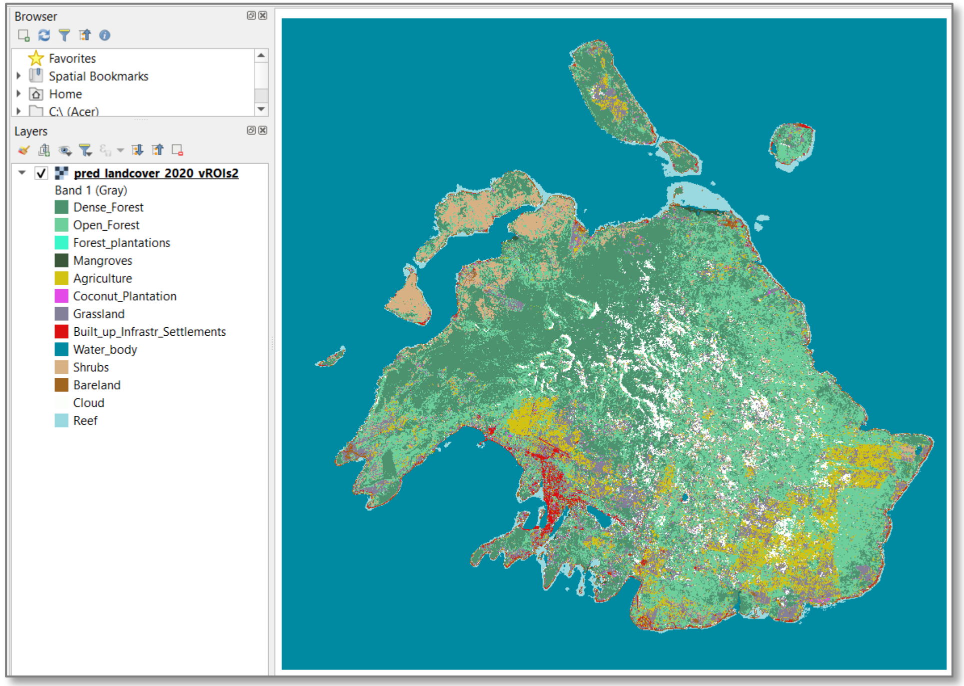
Today – Component 2 (2020)



Today – Your turn

1) Run Component 2 to obtain the land cover classified raster for 2020

- Use the already provided ROIs dataset for 2020 (vROIs2)
- Go through Step 2.1
- Go through Step 2.2
- Go through Step 2.3



Today – Component 2 in R