FORGENIUS

Improving access to FORest GENetic resources Information and services for end-Users

Deliverable D1.5

Method for assessing forest GenRes state through remote-sensing (M48): Published algorithms within an R package to assess forest state using a combination of meteorology and remote-sensing data.

Planned delivery date (as in DoA):M48 31/12/2024
Actual submission date: 06/01/2025, month M49
Workpackage: WP1

Workpackage leader: INRAE

Deliverable leader: JRC

Version: 1.0

Project funded by the European Commission within the Horizon 2020 Programme	
Dissemination Level	
PU Public	
CI Classified, as referred to Commission Decision 2001/844/EC	
CO Confidential, only for members of the consortium (including the Commission Services)	**CO**

Research and Innovation action: GA no. 862221

Start date of the project: January 1st, 2021

TABLE OF CONTENTS

1 Summary

The objective of this deliverable was to develop a methodology to assess the current state of the collection of Genetic Conservation Units (GCUs) through remote sensing. To investigate the spatio-temporal dimensions of climate-related stresses and risks to GCUs, a wide range of data sources, observation techniques and platforms are required. These data are remote-sensing retrievals of land surface temperature, greenness and leaf water content. The remote-sensing data are derived from the outputs of several missions, including data collected by optical and thermal sensors. This package consists of two parts: 1) the extraction of raw remote-sensing data using Google Earth Engine (GEE); and 2) the creation of a cleaned dataset using R (a free software environment for statistical computing). The development of these methods advances the work undertaken as part of Work Package 1 (WP1) by combining data from different sources with the aim of more accurately identifying and assessing the status of GCUs and, in turn, the occurrence of stress events.

2 Introduction

The work undertaken in D1.5 involved the development of a pipeline for downloading, sorting and collating multiple remote sensing-based data sources to assess the status of GCUs. Data extraction was carried out in two main ways: using a multi-petabyte catalogue available via cloud computing platforms (i.e. GEE) and using a dedicated data portal (i.e. R). GEE is used to extract remote-sensing data and R is used to combine GEE extractions. A comprehensive list of data sources is provided in Table 1 (Results section).

2.1 Google Earth Engine GEE

Google Earth Engine (Gorelick et al. 2017) is a cloud-based infrastructure that provides "access to high-performance computing resources for processing very large geospatial datasets". It consists of "a multi-petabyte analysis-ready data catalogue co-located with a high-performance, intrinsically parallel computational service". The data catalogue hosts a repository of geospatial datasets, including the Sentinel 2 optical satellites and the Shuttle Radar Topography Mission (SRTM) map. Google Earth Engine also allows data to be uploaded in raster or vector form. In this study, we uploaded all GCU boundaries as vectors. All data extraction for this study was performed in Google Earth Engine, which provides the ability to compute the zonal statistics of the GCUs and analyse the entire dataset with high computational efficiency.

2.2. R

R is a powerful open-source software environment widely used for statistical computing and data analysis. Originally developed by Ross Ihaka and Robert Gentleman in the 1990s (Ihake et al. 1996), R has evolved into a comprehensive tool supported by a large and active community of users and developers. It is highly extensible, with thousands of packages available through the Comprehensive R Archive Network (CRAN), enabling users to perform advanced statistical modelling, data visualisation and machine learning. R's syntax is user-friendly for both simple and complex data manipulation, making it a popular choice in academia, research and industry. Its integration capabilities with other programming languages and tools further enhance its versatility for data-driven projects.

3 Results

The deliverable consists of a repository of codes and data.

3.1 Codes

There are two codes:

1) Google Earth Engine (GEE) code to extract remote-sensing data

2) R code to combine GEE extractions.

3.1.1.GEE code

GEE code is using Google Colaboratory (Colab). Colab is a free cloud service provided by Google based on the Jupyter Notebook environment and allows to write and execute Python in the browser, with no configuration required, free access to GPUs and easy sharing (Fig. 1). GEE saves results in the Google Drive of the person that launches the code.

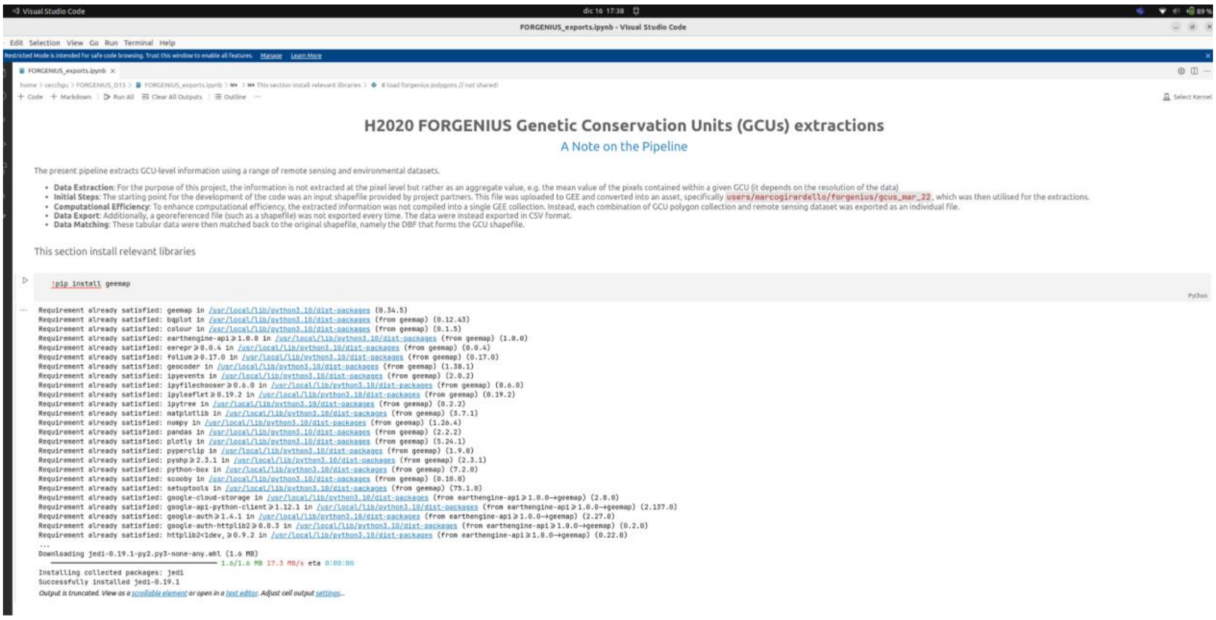

Fig. 1 Screenshot of Google Earth Engine code with COLAB with Visual Studio.

Here is a basic guide on how to use Google Colab for GEE to replicate the analysis:

- **Setting up Colab Notebook** Begin by opening your Google Colab and creating a new Python 3 notebook.
- **Importing Google Earth Engine** To use GEE, you need to import it into your Colab notebook. Use the command import ee. If the library isn't installed, you can install it by using *!pip install earthengine-api*.
- **Authentication** Before you can start using GEE, you need to authenticate your account. This can be done using *ee.Authenticate()*. This line of code will generate a link, click on it, allow access, and finally, copy the generated code and paste it into your Colab notebook. After this step, initialize the library with *ee.Initialize()*.
- **Data Access and Manipulation** With GEE, it is possible to access a vast array of geospatial data and perform various operations on this data, like clipping, mapping, reducing, etc.
- **Visualization** Visualizing the results is an important part of any geospatial analysis. You can use libraries like *folium* or *geemap* to visualize results.

- **Exporting Results** After the analysis, it is possible to export results using *ee.batch.Export.image.toDrive()*. This will save your results to your Google Drive.

The first GEE block refers to static values of topography (Fig.2) and allows to get elevation, slope and aspect of the GCUs.

Topography

Source: Copernicus digital elevation model

This dataset was imported as an asset as it's not available on the GEE public catalogue.

```
# call dem asset i.e. an image collection
dem = ee.ImageCollection("projects/phenology-303215/assets/eu_dem")
elevation = dem.median()

# compute slope and aspects
slope = dem.map(lambda image:ee.Terrain.slope(image)).median()
aspect = dem.map(lambda image:ee.Terrain.aspect(image)).median()

# export elevation data
elevfs = elevation.reduceRegions(collection = fgeniusp, reducer = ee.Reducer.mean().setOutputs(['elevation']),
                       scale = 25)
task = ee.batch.Export.table.toDrive(collection = elevfs,folder = 'Forgenius',
                                     fileFormat = 'CSV',description = 'elevation',
                                     selectors = ['elevation','JRC_ID','EUFGIS_ID'])
task.start()

# export slope data
slopefs = slope.reduceRegions(collection = fgeniusp, reducer = ee.Reducer.mean().setOutputs(['slope']),
                       scale = 25)
task = ee.batch.Export.table.toDrive(collection = slopefs,folder = 'Forgenius',
                                     fileFormat = 'CSV',description = 'slope',
                                     selectors = ['slope','JRC_ID','EUFGIS_ID'])
task.start()


# export aspect data
aspectfs = aspect.reduceRegions(collection = fgeniusp, reducer = ee.Reducer.mean().setOutputs(['aspect']),
                       scale = 25)
task = ee.batch.Export.table.toDrive(collection = aspectfs,folder = 'Forgenius',
                                     fileFormat = 'CSV',description = 'aspect',
                                     selectors = ['aspect','JRC_ID','EUFGIS_ID'])
task.start()
```

Fig. 2 Screenshot of Google Earth Engine code relative to topography.

The second GEE block refers to canopies (Fig.3 to 9) and allows to get: canopy height, dominant leaf type, Land Surface Temperature (LST), Leaf Area Index (LAI), Gross Primary Productivity (GPP), Above Ground Biomass (AGB), Normalized Difference Water Index (NDWI) and Normalized Difference Vegetation Index (NDVI) for each GCUs.

Canopy height

Source: Glad dataset

```
# load collection
TreeH = ee.ImageCollection('users/potapovpeter/GEDI_V27').mosaic()
# apply reducer
c_height = TreeH.reduceRegions(collection = fgeniusp, reducer = ee.Reducer.mean().setOutputs(['canopy_height']),
                   scale = 30)
# export results
task = ee.batch.Export.table.toDrive(collection = c_height, folder = 'Forgenius',
                                     fileFormat = 'CSV',description = 'canopy_height',
                                     selectors = ['canopy_height','JRC_ID','EUFGIS_ID'])
task.start()
```

Dominant Leaf Type

Source: Land Copernicus - High Resolution Layer Dominant Leaf Type

This dataset was imported as an asset as it's not available on the GEE public catalogue.

```
# load image
fortype = ee.Image('users/marcogirardello/forgenius/domleaftypeeu')
fortype = fortype.updateMask(fortype.gt(0))

# apply reducer (mode)
dominant_forest_type = fortype.reduceRegions(collection = fgeniusp,
                                       reducer = ee.Reducer.mode().setOutputs(['dominant_forest_type']),
                                       scale = 12,crs='EPSG:4326')
# export results
task = ee.batch.Export.table.toDrive(collection = dominant_forest_type, folder = 'Forgenius',
                                     fileFormat = 'CSV',description = 'dominant_forest_type',
                                     selectors = ['dominant_forest_type','JRC_ID','EUFGIS_ID'])
task.start()
```

Fig. 3 Screenshot of Google Earth Engine code relative to topography.

Day Land surface temperature aggregated at monthly level.

Catalogue entry: Terra Land Surface Temperature and Emissivity 8-Day Global 1km

```
def lst_filt(img):
    # Select the 'QC_Day' band for quality assessment.
    qa = img.select('QC_Day')

    # Define the cloud bit mask.
    cloudBitMask = (1 << 0)  # Adjust the bit shift as needed for your specific cloud mask.

    # Apply the mask to the image. The bitwiseAnd operation is used to extract the cloud bits,
    # and then eq(0) is used to mask out the pixels where clouds are present.
    mask = qa.bitwiseAnd(cloudBitMask).eq(0)

    # Update the mask of the image, select the 'LST_Day_1km' band,
    # and copy the 'system:time_start' property from the original image.
    return (img.updateMask(mask).
              select('LST_Day_1km').
              copyProperties(img, ['system:time_start']))

# aggregate at monthly level
def year_map(year):
    months = ee.List.sequence(1, 12, 1)
    def month_agg(m):
        month = ee.Number(m)
        filtered_Monthly = lst.filter(ee.Filter.calendarRange(year, year, 'year')) \
                    .filter(ee.Filter.calendarRange(month, month, 'month')) \
                    .mean() \
                    .multiply(0.02)
        return filtered_Monthly.set('year', year) \
          .set('month', m) \
          .set('system:time_start', ee.Date.fromYMD(year, m, 1).millis())
    return months.map(month_agg)

# zonal statistics
def lst_stats(img):
    def set_NA(feature):
        LST = ee.List([feature.get('LST'),-9999]).reduce(ee.Reducer.firstNonNull())
        system_time_start = ee.Image(img).get('system:time_start')
        year = ee.Image(img).get('year')
        month = ee.Image(img).get('month')
        return feature.set({'LST':LST,'imageID':img.id(),
                       'system:time_start':system_time_start,
                       'year':year,
                       'month':month})
    return img.reduceRegions(collection = fgeniusp,
                             reducer = ee.Reducer.mean().setOutputs(['LST']),
                             scale = 926.6254330555,
                             crs = 'SR-ORG:6974').map(set_NA)
```

```
# filter LST collection
lst = (ee.ImageCollection("MODIS/061/MOD11A2").
          filter(ee.Filter.date('2000-01-01', '2021-12-31')).
          select(['LST_Day_1km',"QC_Day"]).
          map(lst_filt))
```

Fig. 4 Screenshot of Google Earth Engine code relative to Land Surface Temperature (LST).

LAI (monthly)

Catalogue entry: [Terra Leaf Area Index/FPAR 8-Day Global 500m]

```
# filter for quality criteria
def lai_qa(img):
    qa = img.select('FparLai_QC')
    cloudBitMask = (1 << 0)
    # cloudBitMask1 = (1 << 0)
    mask = qa.updateMask(qa.eq(0))
    return img.updateMask(mask).select('Lai_500m').copyProperties(img,['system:time_start'])

# aggregate at monthly level
def year_map(year):
    months = ee.List.sequence(1,12,1)
    def month_agg(m):
        month = ee.Number(m)
        filtered_Monthly = lai_filt.filter(ee.Filter.calendarRange(year,year,'year')) \
        .filter(ee.Filter.calendarRange(month,month,'month')) \
        .mean() \
        .multiply(0.1)
        return filtered_Monthly.set('year',year) \
               .set('month',month) \
               .set('system:time_start',ee.Date.fromYMD(year,m,1).millis())
    return months.map(month_agg)

# calculating zonal statistics
def lai_stats(img):
    def set_NA(feature):
        Lai = ee.List([feature.get('Lai'),-9999]).reduce(ee.Reducer.firstNonNull())
        system_time_start = ee.Image(img).get('system:time_start')
        year = ee.Image(img).get('year')
        month = ee.Image(img).get('month')
        return feature.set({'Lai':Lai,'imageID':img.id(),
                          'system:time_start':system_time_start,
                         'year':year,
                         'month':month})
    return img.select('Lai_500m').reduceRegions(collection = fgeniusp,
                                               reducer = ee.Reducer.mean().setOutputs(['Lai']),
                                               scale = 463.3127165275,
                                               crs = 'SR-ORG:6974').map(set_NA)
```

```
# load collection and filter for period of interest and Lai band
lai = (ee.ImageCollection("MODIS/061/MOD15A2H").
         filterDate('2003-01-01','2023-12-31').
         select('Lai_500m','FparLai_QC'))
lai_filt = lai.map(lai_qa)

# aggregate at monthly level
lai_monthly = ee.ImageCollection.fromImages(years.map(year_map).flatten())

# get zonal statistics
lai_statsres = lai_monthly.map(lai_stats).flatten()
```

Fig. 5 Screenshot of Google Earth Engine code relative to Leaf Area Index (LAI).

GPP (monthly)

Catalogue entry: Aqua Gross Primary Productivity 8-Day Global 500m

```
# filter GPP for criteria
def gpp_qa(img):
    qa = img.select('Psn_QC')
    cloudBitMask = 1 << 0
    mask = qa.bitwiseAnd(cloudBitMask).eq(0)
    return img.updateMask(mask) \
      .select("Gpp") \
      .copyProperties(img, ["system:time_start"])

# aggregate at monthly level
def year_map(year):
    months = ee.List.sequence(1, 12, 1)
    def month_agg(m):
        month = ee.Number(m)
        filtered_Monthly = gpp.filter(ee.Filter.calendarRange(year, year, 'year')) \
                    .filter(ee.Filter.calendarRange(month, month.add(0), 'month')) \
                    .mean() \
                    .multiply(0.0001)
        return filtered_Monthly.set('year', year) \
          .set('month', m) \
          .set('system:time_start', ee.Date.fromYMD(year, m, 1).millis())
    return months.map(month_agg)

# zonal statistics
def gpp_stats(img):
    def set_NA(feature):
        Gpp = ee.List([feature.get('Gpp'),-9999]).reduce(ee.Reducer.firstNonNull())
        system_time_start = ee.Image(img).get('system:time_start')
        year = ee.Image(img).get('year')
        month = ee.Image(img).get('month')
        return feature.set({'Gpp':Gpp,'imageID':img.id(),
                       'system:time_start':system_time_start,
                       'year':year,
                       'month':month})
    return img.select('Gpp').reduceRegions(collection = fgeniusp,
                             reducer = ee.Reducer.mean().setOutputs(['Gpp']),
                             scale = 463.3127165275,
                             crs = 'SR-ORG:6974').map(set_NA)
```

```
gpp = (ee.ImageCollection("MODIS/006/MYD17A2H").
              filterDate('2003-01-01','2021-12-31'))

gpp = gpp.map(gpp_qa)

# aggregate to a monthly resolution
gpp_monthly = ee.ImageCollection.fromImages(years.map(year_map).flatten())

# calculate zonal statistics
gpp_zonstats = gpp_monthly.map(gpp_stats).flatten()
```

Fig. 6 Screenshot of Google Earth Engine code relative to Gross Primary Productivity (GPP).

Biomass for the year 2010

Source: ESA Biomass datasets

```
EU_BIOMASS = ee.Image("users/guidolavespa2511/EU_biomass")

biostats = EU_BIOMASS.reduceRegions(collection = fgeniusp,reducer = ee.Reducer.mean().setOutputs(['biomass_2010']),
                         scale = 98.95114197446169,crs = 'EPSG:4326')

task = ee.batch.Export.table.toDrive(collection = biostats, folder = 'Forgenius',fileFormat = 'CSV',
                                    description = 'biomass_2010')
task.start()
```

Fig. 7 Screenshot of Google Earth Engine code relative to Above Ground Biomass (AGB).

NDWI

Catalogue entry: MODIS Combined 16-Day NDWI

```
# function for aggregating data at monthly resolution
years = ee.List.sequence(2003, 2021, 1)

def year_map(year):
    months = ee.List.sequence(1, 12, 1)
    def month_agg(m):
        month = ee.Number(m)
        filtered_Monthly = ndwi.filter(ee.Filter.calendarRange(year, year, 'year')) \
                    .filter(ee.Filter.calendarRange(month, month.add(0), 'month')) \
                    .mean()
        return filtered_Monthly.set('year', year) \
          .set('month', m) \
          .set('system:time_start', ee.Date.fromYMD(year, m, 1).millis())
    return months.map(month_agg)

# function for zonal statistics
def ndwi_stats(img):
    def set_NA(feature):
        NDWI = ee.List([feature.get('NDWI'),-9999]).reduce(ee.Reducer.firstNonNull())
        system_time_start = ee.Image(img).get('system:time_start')
        year = ee.Image(img).get('year')
        month = ee.Image(img).get('month')
        return feature.set({'NDWI':NDWI,'imageID':img.id(),
                       'system:time_start':system_time_start,
                       'year':year,
                       'month':month})
    return img.select('NDWI').reduceRegions(collection = fgeniusp,
                                   reducer = ee.Reducer.mean().setOutputs(['NDWI']),
                                   scale = 463.3127165275,
                                   crs = 'SR-ORG:6974').map(set_NA)
```

```
ndwi = ee.ImageCollection("MODIS/MCD43A4_006_NDWI").filterDate('2001-01-01','2021-12-31')

# aggregate to a monthly resolution
ndwi_monthly = ee.ImageCollection.fromImages(years.map(year_map).flatten())

# calculate zonal statistics
ndwi_zonstats = ndwi_monthly.map(ndwi_stats).flatten()
```

```
# export data
task = ee.batch.Export.table.toDrive(collection = ndwi_zonstats, folder = 'Forgenius',
                                      fileFormat = 'CSV',description = 'ndwi',selectors = ['NDWI','JRC_ID','EUFGIS_ID','month','year'])
task.start()
```

Fig. 8 Screenshot of Google Earth Engine code relative to Normalized Difference Water Index (NDWI).

NDVI from Sentinel 2

Catalogue entry: [Harmonized Sentinel-2 MSI: MultiSpectral Instrument, Level-1C]

```python
# NDVI calculation
def add_ndvi(img):
    ndvi = img.normalizedDifference(['B8','B4']).multiply(1000).rename('ndvi')
    return img.addBands(ndvi)

# rescaling bands
def rescale(img, exp, thresholds):
    return img.expression(exp, {'img': img}) \
        .subtract(thresholds[0]).divide(thresholds[1] - thresholds[0])

# cloudscoring
def sentinelCloudScore(img):
    # Compute several indicators of cloudyness and take the minimum of them.
    score = ee.Image(1)
    # Clouds are reasonably bright in the blue and cirrus bands.
    score = score.min(rescale(img, 'img.blue', [0.1, 0.5]))
    score = score.min(rescale(img, 'img.cb', [0.1, 0.3]))
    score = score.min(rescale(img, 'img.cb + img.cirrus', [0.15, 0.2]))

    # Clouds are reasonably bright in all visible bands.
    score = score.min(rescale(img, 'img.red + img.green + img.blue', [0.2, 0.8]))

    # Clouds are moist
    ndmi = img.normalizedDifference(['nir','swir1'])
    score=score.min(rescale(ndmi, 'img', [-0.1, 0.1]))

    # However, clouds are not snow
    ndsi = img.normalizedDifference(['green', 'swir1'])
    score=score.min(rescale(ndsi, 'img', [0.8, 0.6]))
    score = score.multiply(100).byte()
    return img.addBands(score.rename('cloudScore'))

# mask clouds using the Sentinel-2 QA band
def maskS2clouds(img):
    qa = img.select('QA60').int16()
    # bits 10 and 11 are cloud and cirrus, respectively
    cloudBitMask = 1 << 10
    cirrusBitMask = 1 << 11
    # both flags should be set to zero, indicating clear conditions
    mask = qa.bitwiseAnd(cloudBitMask).eq(0) \
    .And(qa.bitwiseAnd(cirrusBitMask))
    # return the masked and scaled data
    return img.updateMask(mask).copyProperties(img,['system:time_start'])

# merge bands
def mergeImageBands(joinResult):
    return ee.Image(joinResult.get('primary')) \
           .addBands(joinResult.get('secondary'))

# only select relevant bands
def clean(img):
    # rescale 0-1
    t = img.select([ 'B1','B2','B3','B4','B5','B6','B7',
                    'B8','B8A', 'B9','B10', 'B11','B12']).divide(10000)
    # add QA60 and probability bands
    t = t.addBands(img.select(['QA60','probability']))
    out = t.copyProperties(img).copyProperties(img,['system:time_start'])
```

Fig. 9 Screenshot of Google Earth Engine code relative to Normalized Difference Vegetation Index (NDVI).

3.1.2.R code

R code consists of a script *extraction_wrapup_0322.R* that ingests the output of GEE extraction and generates the output files (Fig. 10).

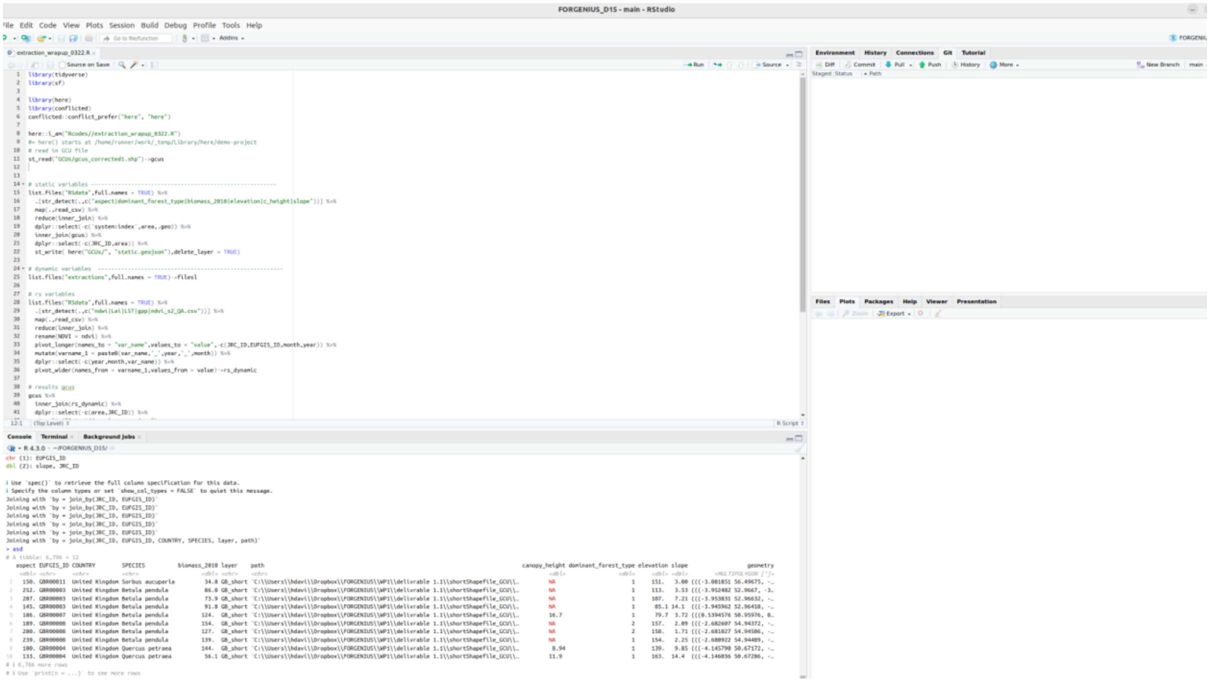

Fig. 10 Screenshot of R code to generate output.

3.2 Data

The output data consists of two *geojson* files containing time series data for 6 static and 5 dynamic indicators (Table 1).

The two output files are:

1) *static.geojson*

contains information on canopy height, dominant leaf type, Above Ground Biomass (AGB), elevation, slope and aspect.

2) *dynamic_rs.geojson*

contains information on Land Surface Temperature (LST), Leaf Area Index (LAI), Gross Primary Productivity (GPP), Normalized Difference Water Index (NDWI) and Normalized Difference Vegetation Index (NDVI).

A list of the remote-sensing variables extracted for each polygon, within each GCU, is described in Table 1.

Variables are grouped into categories (Variable group column), depending on the broad physical or biological properties they refer to. The dataset original sources have been reported in the form of a URL and they were last accessed on 29/11/2024.

Type of indicator	Variable	Period	Data source	Category
Topography	Elevation	2011	**https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1**	**Static**
Topography	Slope	2011		
Topography	Aspect	2011		
Structural properties of the vegetation	Tree height	2019-2021	https://glad.umd.edu/dataset/gedi	
Structural properties of the vegetation	Above Ground Biomass (AGB)	2010	https://globbiomass.org/	
Land cover	Dominant leaf type (conifers vs broadleaf)	2018	**https://land.copernicus.eu/pan-european/high-resolution-layers/forests/dominant-leaf-type**	
Structural properties of the vegetation	Leaf Area Index (LAI)	2003-2021	https://lpdaac.usgs.gov/products/mod15a2hv006/	**Dynamic**
Structural properties of the vegetation	Gross Primary Productivity (GPP)	2003-2021	https://lpdaac.usgs.gov/products/mod17a2hv006/	
Physiological indexes, water/thermal stress indices	Normalized Difference Water Index (NDWI)	2001-2021	https://developers.google.com/earth-engine/datasets/catalog/MODIS_MCD43A4_006_NDWI?hl=en	
Physiological indexes, water/thermal stress indices	Normalized Difference Vegetation Index (NDVI)	2016-2021	https://scihub.copernicus.eu/	

Table 1. Simplified list of the remote-sensing based indicators provided as part of the deliverable. All the data are provided at the GCU-level.

3.3 Repository Description

The GIT repository is organized as follows:

Root Files

- FORGENIUS_D15.Rproj

RStudio project file, which serves as the entry point to the repository when opened in RStudio.

- FORGENIUS_exports.ipynb

A Jupyter Notebook file, likely containing data export workflows, visualizations, or analysis. Note that the file with the location of GCU is not shared to prevent unauthorized accesses.

-.gitignore

Configuration file specifying which files or directories should be ignored by Git version control (e.g., temporary or output files).

Directories

- extractions/

Contains intermediate outputs related to extraction processes with GEE.

- GCUs/

A directory that stores the GCUs shapefile.

- Output/

Holds output files, such as results, generated from R script/workflow.

- Rcodes/

Contains R script files for various analyses, processes, or computations. This is the main folder for R code.

- RSdata/

A directory for raw or processed datasets used by the R scripts, organized in .csv or .RData formats.

- .git/

The Git directory managing version control metadata.

The repository is designed for GEE/R-based data analysis workflows, involving:

RStudio projects for code organization and reproducibility.

Jupyter Notebooks for interactive analysis or export operations.

Version control with Git for collaborative work and tracking changes.

Well-organized subdirectories to manage code, data, and outputs separately.

4 Conclusions

We have successfully developed and implemented a pipeline designed to extract and collate a wide range of remote-sensing data. This has enabled us to assess the status of the GCUs and, in turn, effectively derive a range of heat and drought indicators. Our framework serves as a solid foundation for establishing an operational framework aimed at regularly extracting these remotely sensed indicators for the GCU collection.

5 Partners involved in the work

Data collation analysis was led by JRC. INRAE collected and harmonised the geographical data of the GCUs (polygons describing the contours). CREAF contributed to the identification of appropriate data sources.

6 Annexes

Annex A1: A zip file containing CSV and geojson files with individual meteorology and remote-sensing data.
GitHub Repository: https://github.com/guidoceccherini/FORGENIUS_D15

References

Gorelick, N. et al. Google Earth Engine: planetary-scale geospatial analysis for everyone. Remote Sens. Environ. 202, 18–27 (2017).

Ihaka, R., and Gentleman R. “R: A Language for Data Analysis and Graphics.” Journal of Computational and Graphical Statistics 5, no. 3 (1996).