Downloading Clipped Satellite data using Google Earth Engine

Aim

The aim of this Complex Engineering Problem is to devise a tool which lets user to download Landsat data or any other data that is available in Google Earth library of a specific ROI (Region of Interest).

Introduction

Landsat satellite imagery is useful for a variety of purposes, including land cover research, environmental monitoring, and change detection. Accessing and downloading Landsat data for a specific area, on the other hand, can be a difficult operation, especially for individuals unfamiliar with remote sensing and programming. This report tackles this issue by providing a snippet of Python code that automates the process of obtaining Landsat images inside a specific Region Of Interest.

Solution

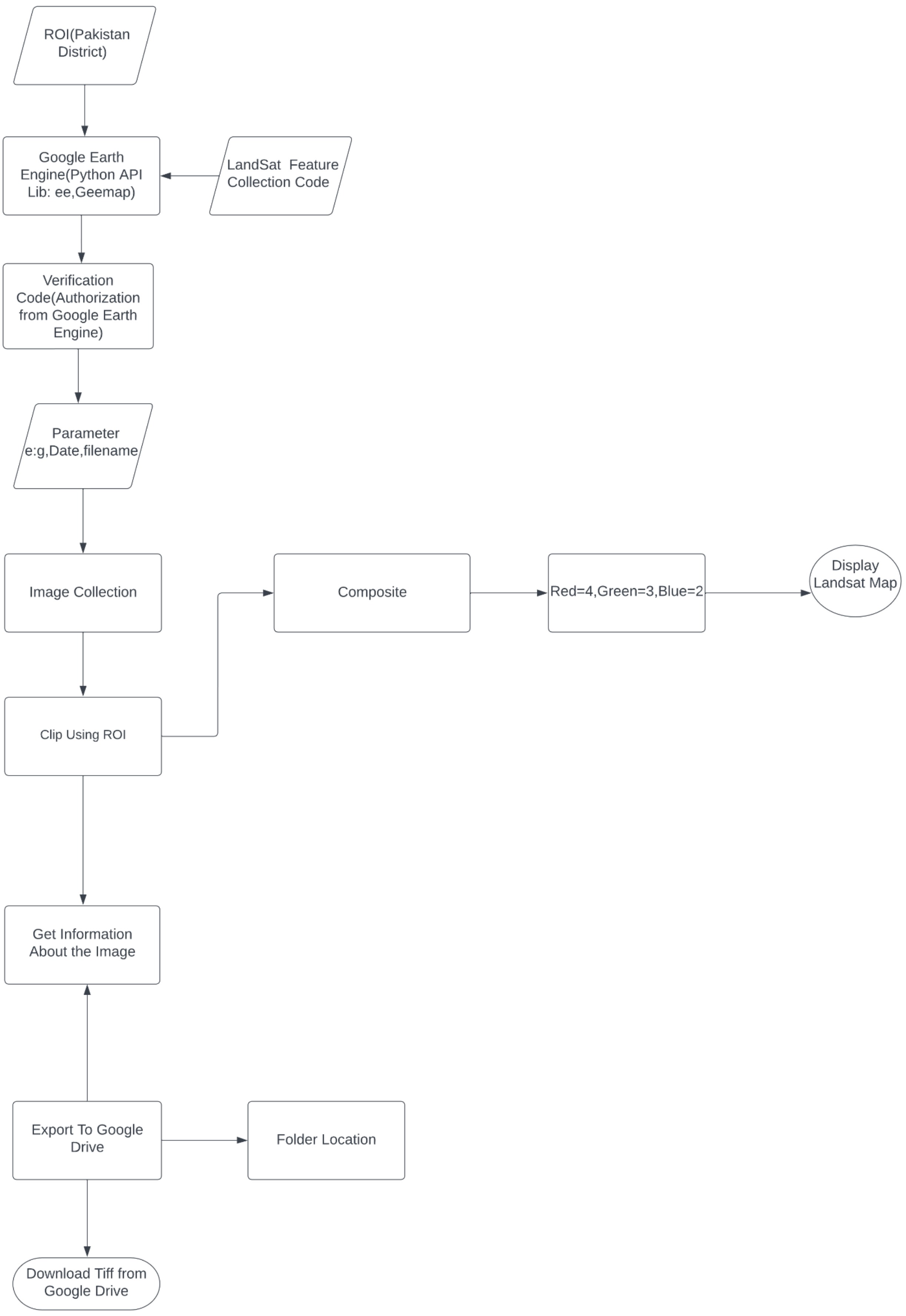
The solution entails using the Earth Engine (EE) platform and the geemap package, which makes working with Earth Engine in Python simpler. Users can set the ROI using a district name, start and end dates for data gathering, and a folder to store the downloaded image in the code snippet. The Landsat image collection is filtered by ROI and time period, and a composite image is constructed from the Landsat data. After clipping the composite image to the ROI, the desired bands are picked for visualisation. Finally, the clipped image is saved to Google Drive via the batch export capabilities of Earth Engine.

Requirements

I. Python IDE (Jupyter Notebook recommended) II.
Google Earth Engine Account

Workflow

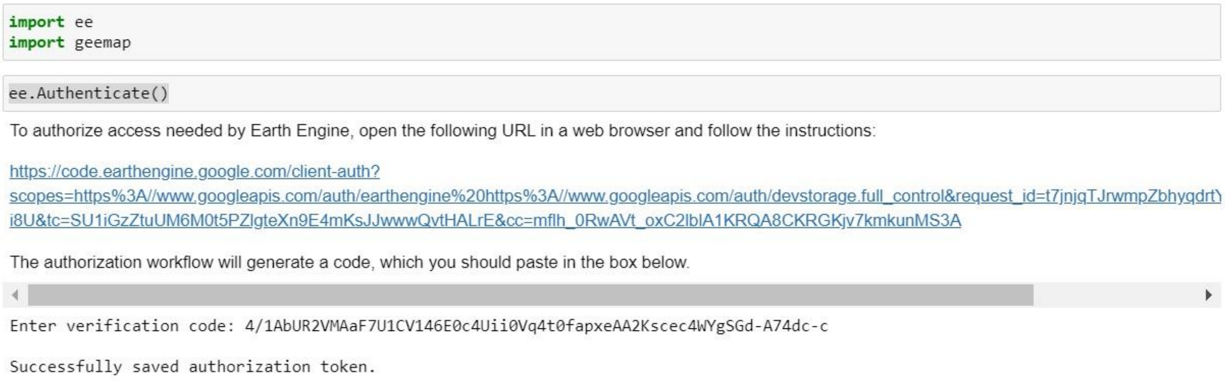
A graphical representation of the methodology used to develop this tool is demonstrated below using a flowchart.


ROI(Pakistan District)
Google Earth Engine(Python API Lib: ee,Geemap)
LandSat Feature Collection Code
Verification Code(Authorization from Google Earth Engine)
Parameter e:g,Date,filename
Image Collection
Composite
Red=4,Green=3,Blue=2
Display Landsat Map
Clip Using ROI
Get Information About the Image
Export To Google Drive
Folder Location
Download Tiff from Google Drive

Method

The following is a detailed description of the approach for obtaining Landsat data inside a given Region of Interest (ROI) in Jupyter Notebook using the Earth Engine python API:

I. <u>Authentication and Initialization</u>: The process begins with the Earth Engine (EE) library being authenticated and initialised. This step guarantees that the user has access to the Earth Engine platform and the relevant permissions.



```
import ee
import geemap
```

```
ee.Authenticate()
```

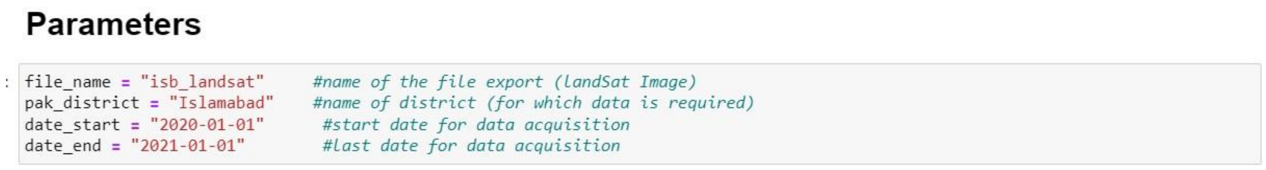
To authorize access needed by Earth Engine, open the following URL in a web browser and follow the instructions:

https://code.earthengine.google.com/client-auth?
scopes=https%3A//www.googleapis.com/auth/earthengine%20https%3A//www.googleapis.com/auth/devstorage.full_control&request_id=t7jnjqTJrwmpZbhyqdrt
i8U&tc=SU1iGzZtuUM6M0t5PZlgteXn9E4mKsJJwwwQvtHALrE&cc=mflh_0RwAVt_oxC2lblA1KRQA8CKRGKjv7kmkunMS3A

The authorization workflow will generate a code, which you should paste in the box below.

```
Enter verification code: 4/1AbUR2VMAaF7U1CV146E0c4Uii0Vq4t0fapxeAA2Kscec4WYgSGd-A74dc-c

Successfully saved authorization token.
```

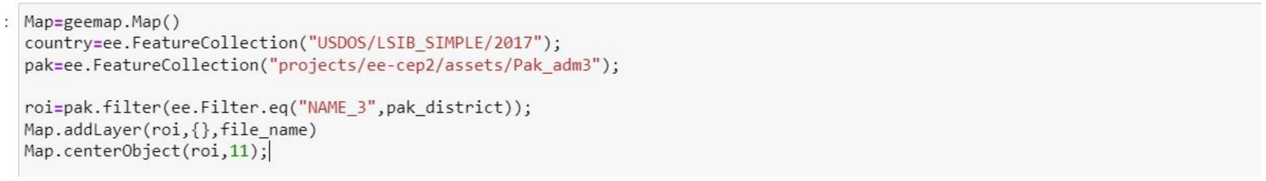
II. <u>Defining Parameters:</u> The user specifies a number of parameters that are required for downloading Landsat data. The file_name (name of the exported file), pak_district (name of the district for which data is requested), date_start (start date for data acquisition), and date_end (end date for data acquisition) are among the parameters. These characteristics can be tailored to the needs of the user.



Parameters

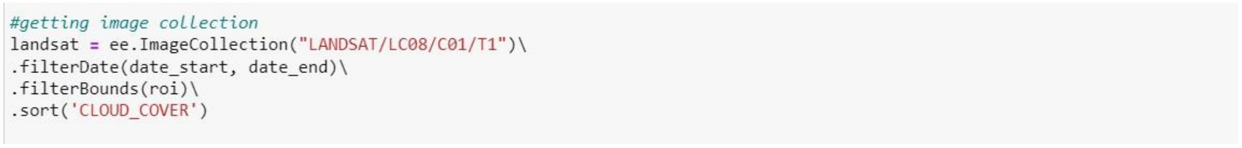
```
file_name = "isb_landsat"      #name of the file export (landSat Image)
pak_district = "Islamabad"     #name of district (for which data is required)
date_start = "2020-01-01"       #start date for data acquisition
date_end = "2021-01-01"         #last date for data acquisition
```

III. <u>Loading Feature Collections and Creating ROI:</u> The script loads the feature collections required to calculate the ROI. The USDOS/LSIB_SIMPLE/2017 collection contains country-level boundaries in this example, while the Pak_adm3 collection contains administrative boundaries for Pakistan which was uploaded as a Asset. To extract the specified ROI for the Landsat data, the script filters the administrative boundaries based on the pak_district argument.



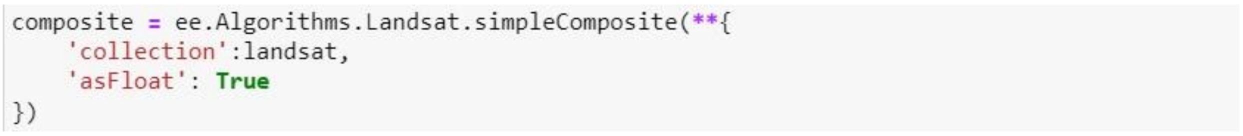
```
Map=geemap.Map()
country=ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017");
pak=ee.FeatureCollection("projects/ee-cep2/assets/Pak_adm3");

roi=pak.filter(ee.Filter.eq("NAME_3",pak_district));
Map.addLayer(roi,{},file_name)
Map.centerObject(roi,11);
```

IV. <u>Loading the Landsat Image Collection:</u> The ee.ImageCollection function is used to load the Landsat image collection. The Landsat 8 Surface Reflectance (C01) Tier 1 collection (LANDSAT/LC08/C01/T1) is specified in the script. The collection is then filtered using the filterBounds() function based on the ROI, ensuring that only photographs within the stated ROI are included.



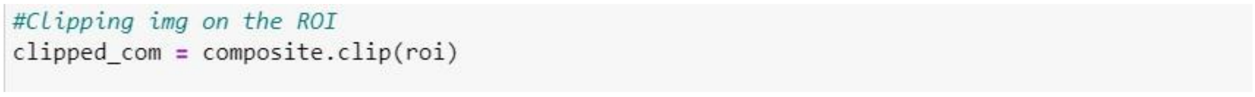
```
#getting image collection
landsat = ee.ImageCollection("LANDSAT/LC08/C01/T1")\
.filterDate(date_start, date_end)\
.filterBounds(roi)\
.sort('CLOUD_COVER')
```

V. <u>Creating a Composite Image:</u> The script uses the ee.Algorithms.Landsat.simpleComposite() function to create a composite image from the Landsat image collection. This function generates a composite image by mixing various photographs from the collection and taking into account cloud cover and other variables. The resulting composite image is a single image with less cloud cover and higher image quality.



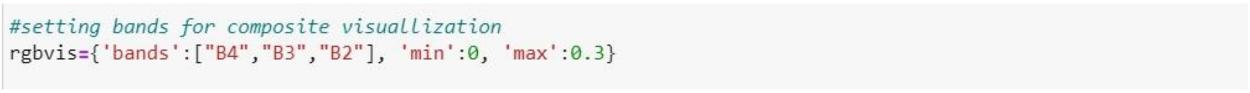
```
composite = ee.Algorithms.Landsat.simpleComposite(**{
    'collection':landsat,
    'asFloat': True
})
```

VI. <u>Clipping the Composite Image to the ROI:</u> The following step is to clip the composite image to the defined ROI. This is accomplished by the use of the clip() function, which limits the image extent to the bounds of the ROI. The clip() function is used to the composite image, retaining only the appropriate region within the ROI.



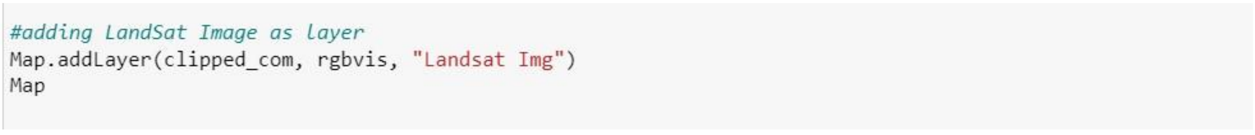
```
#Clipping img on the ROI
clipped_com = composite.clip(roi)
```

VII. <u>Defining Visualisation Parameters</u>: The script defines visualisation parameters in order to visualise the Landsat picture on the map. In this scenario, the script visualises the Red, Green, and Blue (B4, B3, B2) bands. The rgbvis variable provides the bands to be displayed as well as the visualization's minimum and maximum values.



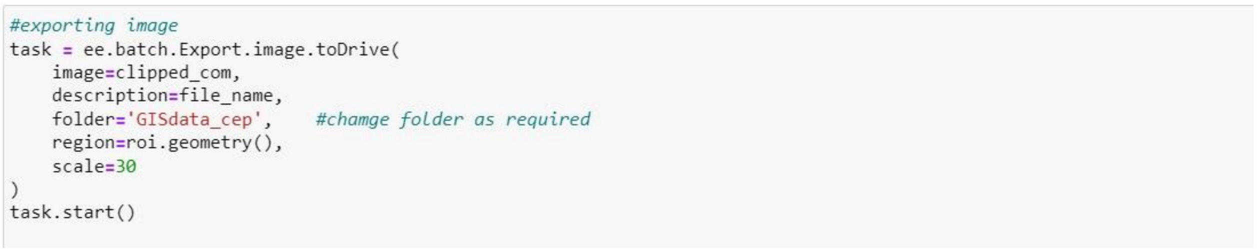
```
#setting bands for composite visuallization
rgbvis={'bands':["B4","B3","B2"], 'min':0, 'max':0.3}
```

VIII. <u>Adding the Landsat Clipped Image to the Map:</u> In Jupyter Notebook, the geemap library is used to build an interactive map. Using the addLayer() function, the script adds the clipped Landsat image as a layer to the map. Within the defined ROI, the map displays a visual depiction of the Landsat image.



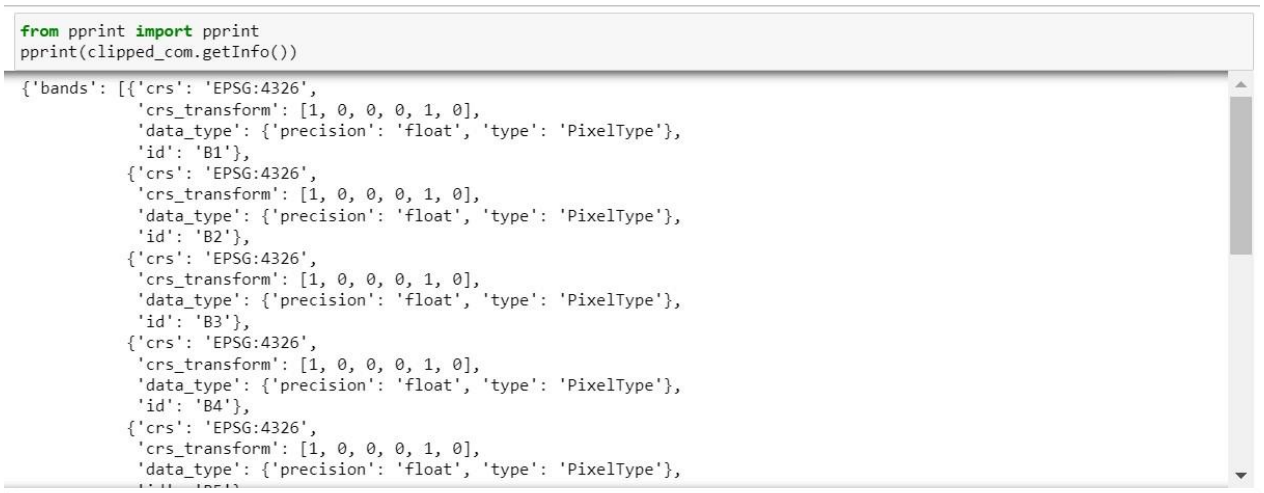
```
#adding LandSat Image as layer
Map.addLayer(clipped_com, rgbvis, "Landsat Img")
Map
```

IX. <u>Exporting the Image:</u> The script uses the Earth Engine batch export functionality to download the Landsat image. The cropped picture is exported to Google Drive using the ee.batch.Export.image.toDrive() function. The image to export, a description for the produced file, the folder in Google Drive for storage, the scale (spatial resolution) of the exported image, and the region (ROI geometry) to export are all parameters passed to the function. These parameters are tailored to the defined ROI and desired outcome.



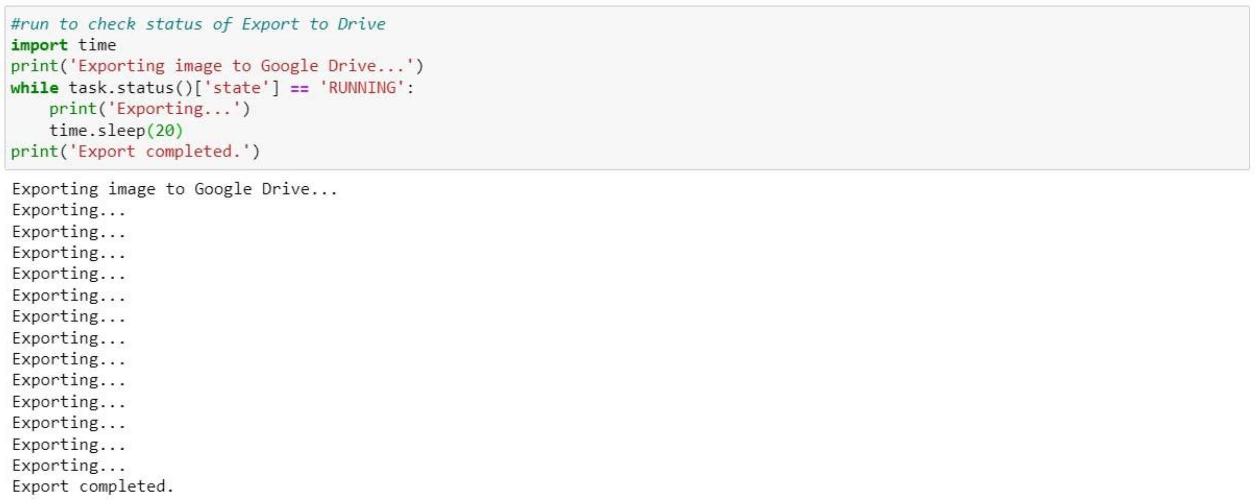
```
#exporting image
task = ee.batch.Export.image.toDrive(
    image=clipped_com,
    description=file_name,
    folder='GISdata_cep',    #chamge folder as required
    region=roi.geometry(),
    scale=30
)
task.start()
```

X. <u>Monitoring the Export Progress:</u> After starting the export , the script uses a while loop to continuously check the task's status. At regular intervals, the task's condition is checked. This guarantees that the user is kept up to date on the status of the export.



```
from pprint import pprint
pprint(clipped_com.getInfo())
```

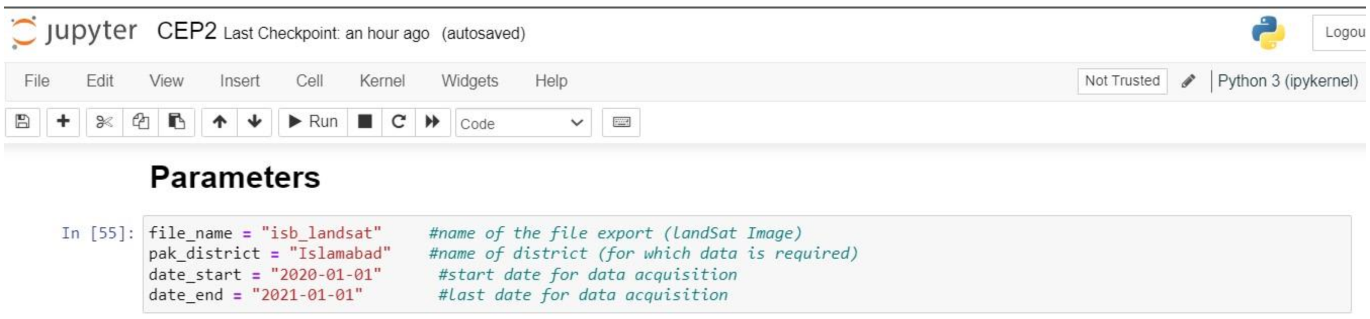
```
{'bands': [{'crs': 'EPSG:4326',
            'crs_transform': [1, 0, 0, 0, 1, 0],
            'data_type': {'precision': 'float', 'type': 'PixelType'},
            'id': 'B1'},
           {'crs': 'EPSG:4326',
            'crs_transform': [1, 0, 0, 0, 1, 0],
            'data_type': {'precision': 'float', 'type': 'PixelType'},
            'id': 'B2'},
           {'crs': 'EPSG:4326',
            'crs_transform': [1, 0, 0, 0, 1, 0],
            'data_type': {'precision': 'float', 'type': 'PixelType'},
            'id': 'B3'},
           {'crs': 'EPSG:4326',
            'crs_transform': [1, 0, 0, 0, 1, 0],
            'data_type': {'precision': 'float', 'type': 'PixelType'},
            'id': 'B4'},
           {'crs': 'EPSG:4326',
            'crs_transform': [1, 0, 0, 0, 1, 0],
            'data_type': {'precision': 'float', 'type': 'PixelType'},
```



```
#run to check status of Export to Drive
import time
print('Exporting image to Google Drive...')
while task.status()['state'] == 'RUNNING':
    print('Exporting...')
    time.sleep(20)
print('Export completed.')
```

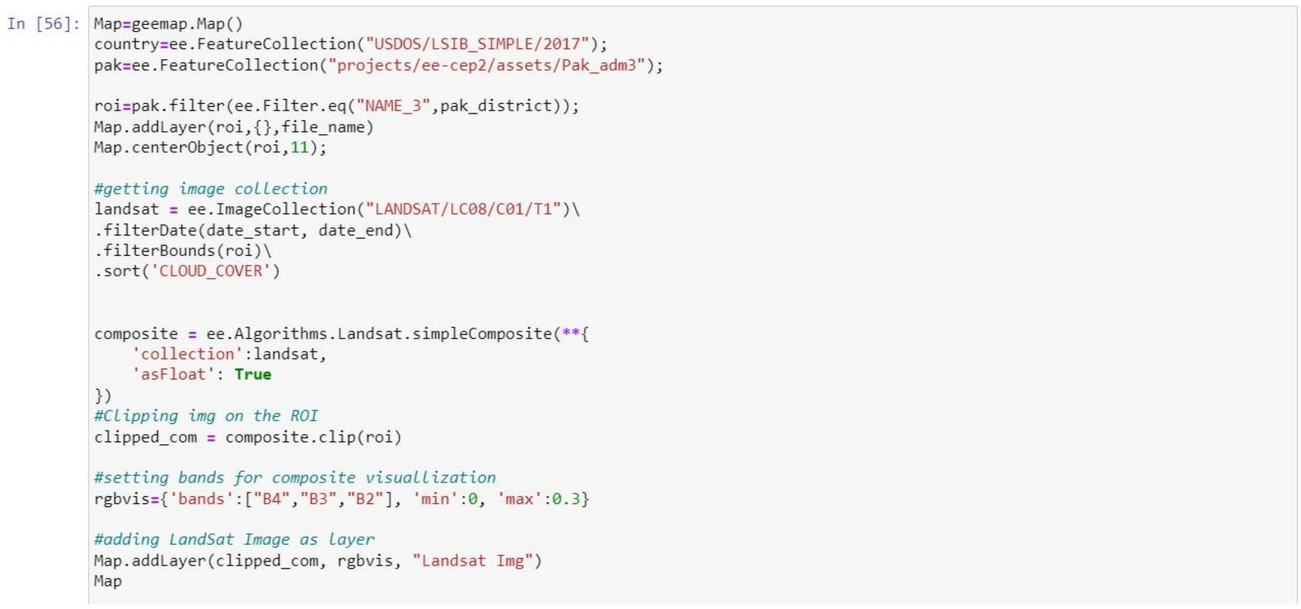
```
Exporting image to Google Drive...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Exporting...
Export completed.
```

User Interface


jupyter CEP2 Last Checkpoint: an hour ago (autosaved)
Logou
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

Parameters

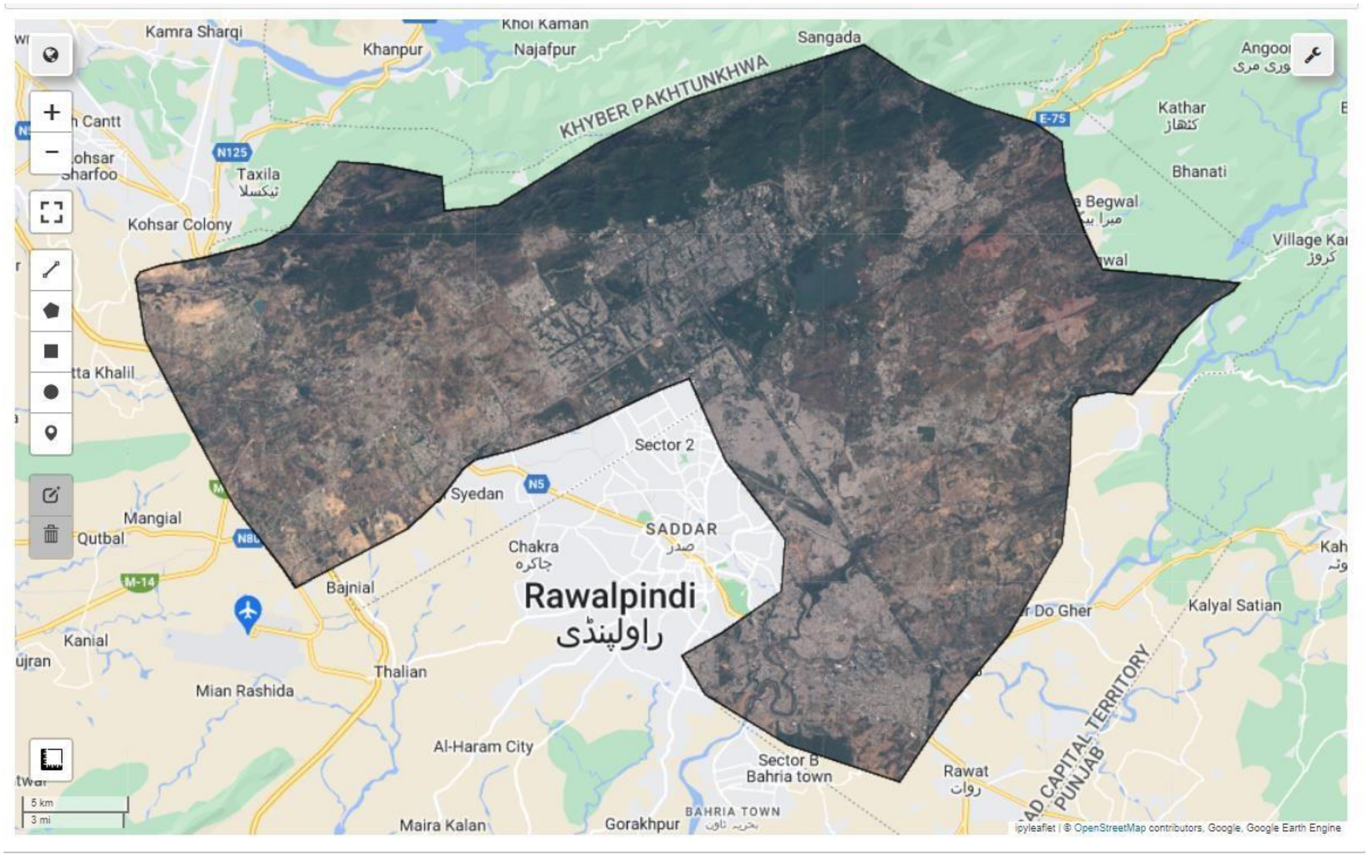
```
In [55]: file_name = "isb_landsat"      #name of the file export (landSat Image)
         pak_district = "Islamabad"     #name of district (for which data is required)
         date_start = "2020-01-01"       #start date for data acquisition
         date_end = "2021-01-01"         #last date for data acquisition
```



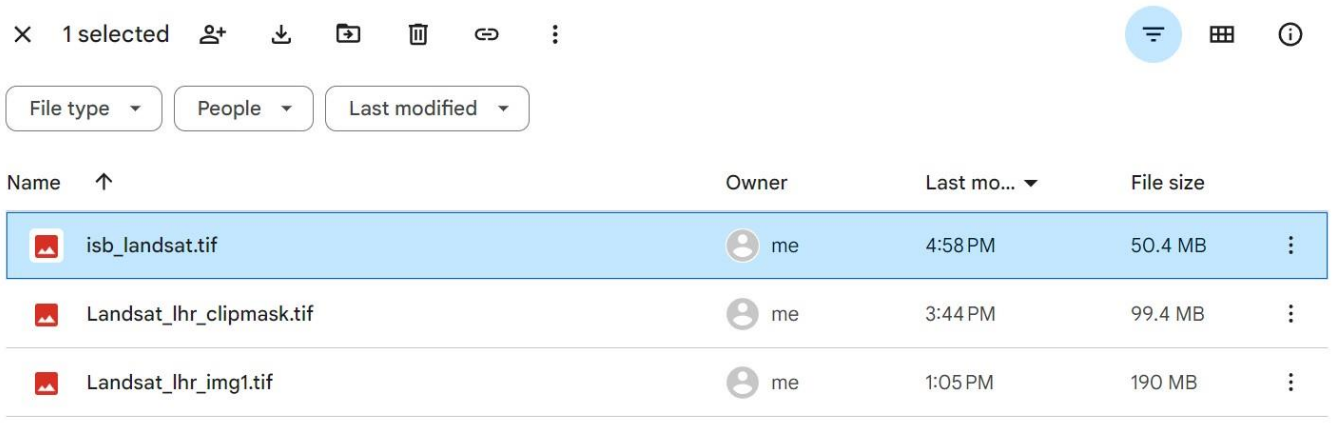
```
In [56]: Map=geemap.Map()
         country=ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017");
         pak=ee.FeatureCollection("projects/ee-cep2/assets/Pak_adm3");

         roi=pak.filter(ee.Filter.eq("NAME_3",pak_district));
         Map.addLayer(roi,{},file_name)
         Map.centerObject(roi,11);

         #getting image collection
         landsat = ee.ImageCollection("LANDSAT/LC08/C01/T1")\
         .filterDate(date_start, date_end)\
         .filterBounds(roi)\
         .sort('CLOUD_COVER')


         composite = ee.Algorithms.Landsat.simpleComposite(**{
             'collection':landsat,
             'asFloat': True
         })
         #Clipping img on the ROI
         clipped_com = composite.clip(roi)

         #setting bands for composite visuallization
         rgbvis={'bands':["B4","B3","B2"], 'min':0, 'max':0.3}

         #adding LandSat Image as layer
         Map.addLayer(clipped_com, rgbvis, "Landsat Img")
         Map
```

Results

Map showing ROI boundary and landsat image.

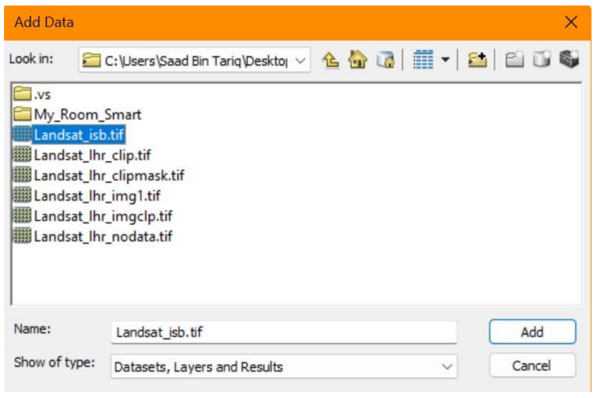


Exported Tif file in Google Drive

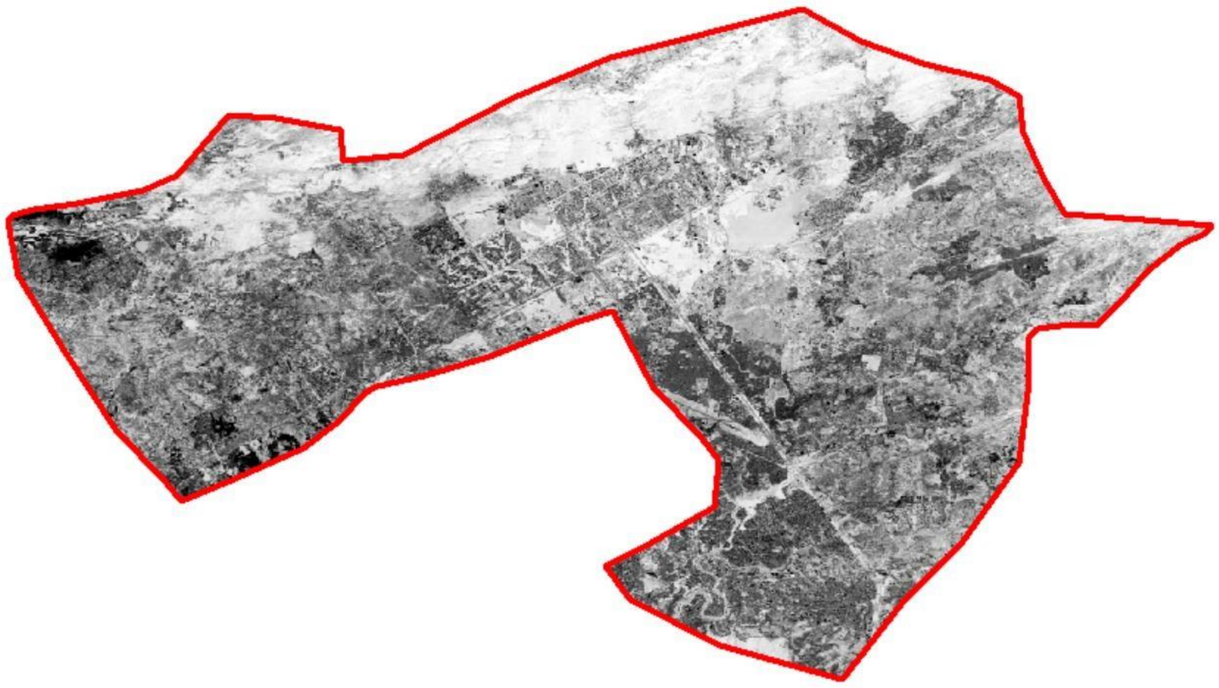


Name	Owner	Last mo...	File size
isb_landsat.tif	me	4:58 PM	50.4 MB
Landsat_lhr_clipmask.tif	me	3:44 PM	99.4 MB
Landsat_lhr_img1.tif	me	1:05 PM	190 MB

Opening Landsat Image in ArcMap



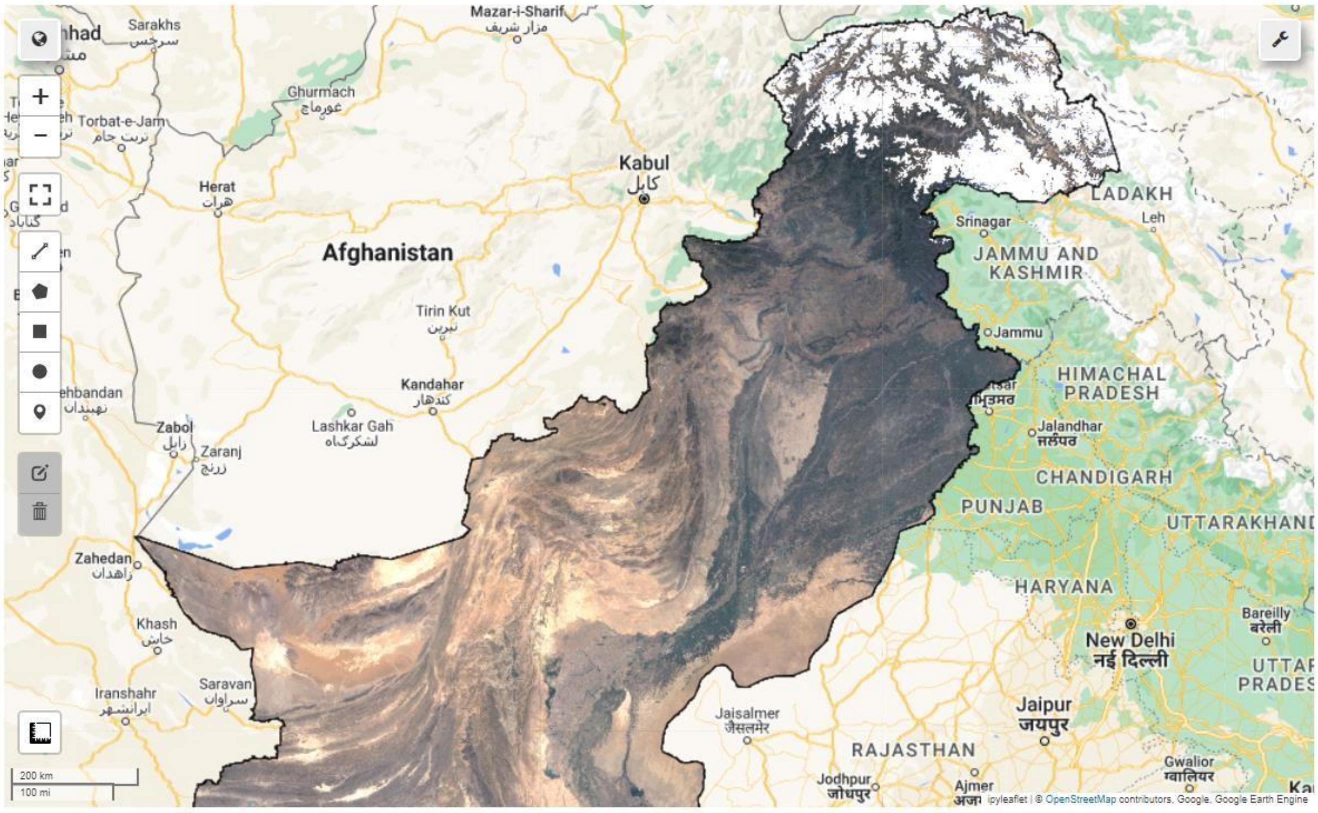
Band 4 is Added



Landsat Imagery of Pakistan

```
ee.Initialize()
```

```
Map=geemap.Map()
country=ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017");
roi=country.filter(ee.Filter.eq("country_na","Pakistan"));
Map.addLayer(roi,{},"Pak")
Map.centerObject(roi,8);


landsat = ee.ImageCollection("LANDSAT/LC08/C01/T1")\
.filterDate('2020-01-01', '2021-01-01')\
.filterBounds(roi)
composite = ee.Algorithms.Landsat.simpleComposite(**{
    'collection':landsat,
    'asFloat': True
})
rgbvis={'bands':["B4","B3","B2"], 'min':0, 'max':0.3}
Map.addLayer(composite.clip(roi), rgbvis, "Landsat Img")
Map
```



Steps to Download Landsat Imagery

Follow these steps to use the code snippet for downloading Landsat data within a defined ROI:

I. Make sure you have the Python libraries ee and geemap installed.
II. Run the ee (Library) to authenticate and initialise the Earth Engine.Authenticate() and ee are two Initialise() functions.
III. Copy and paste the provided code into a Python script or Jupyter Notebook.
IV. Change the variables file_name, pak_district, date_start, and date_end to meet your needs. V. Execute the code snippet.
VI. Keep an eye on the output to see how far the export task has progressed.

VII. The Landsat image will be downloaded to the selected location in your Google Drive once the activity is completed.

Advantages

The designed tool has various benefits:

I. Makes it easier to obtain Landsat data for a specific ROI.
II. Filtering, trimming, and exporting the image are all automated.
III. Users can customise the ROI, time range, and image storage folder.
IV. Allows you to visualise and export Landsat imagery depending on particular bands. V. Google Drive integration for simple access

Difficulties faced during development

Several difficulties were encountered during the development process:

I. Comprehending and putting the Earth Engine API and geemap library into action. II. Filtering the Landsat image collection by ROI and time interval.
III. Any satellite dataset available on Google Earth Engine can be downloaded
IV. Configuring the export parameters and monitoring the status of the export task. V. Assuring that the Earth Engine is properly authenticated and initialised.

Conclusion

In a nutshell, this tool, which makes use of the Python EE library and geemap, offers a streamlined and automated way for obtaining Landsat data inside a given Region of Interest (ROI). The technology automates the acquisition of satellite imagery by eliminating the requirement for human image filtering and clipping. Users can quickly download Landsat pictures for their preferred area and time period by defining the ROI, start and end dates, and a folder for storage.

The Jupyter Notebook code snippet walks through the process step by step, including authentication, loading the Landsat image collection, producing a composite image, clipping to the ROI, and exporting the image to Google Drive. By visualising the ROI and the downloaded Landsat image on an interactive map, the geemap library improves the user experience.

The benefits of using this technology are substantial. It makes it easier to obtain Landsat data, especially for those who are unfamiliar with remote sensing and programming. Users can focus on the study and interpretation of the acquired satellite images by automating the steps of filtering, cutting, and exporting the image.