ABSTRACT

Over the past few years, Earth Observation (EO) has been continuously generating much spatio-temporal data that serves for societies in resource surveillance, environment protection, and disaster prediction. The proliferation of EO data poses great challenges in current approaches for data management and processing. Nowadays, the Array Database technologies show great promise in managing and processing EO Big Data. This project suggests storing and processing EO data as multidimensional arrays based on state-of-the-art array database technologies. A multidimensional spatiotemporal array model is proposed for EO data with specific strategies for mapping spatial coordinates to dimensional coordinates in the model transformation. It allows consistent query semantics in databases and improves the database computing by adopting unified array models in databases for EO data. Our approach is implemented as an extension to Open Data Cube, an open-source array database software and at final step, there is an Open Data Cube UI through which the stored multidimensional data can be retrieved, viewed and can apply queries to view the desired results.

Table of Contents

List of Tables

List of Figures

CHAPTER-1

INTRODUCTION

1.1 Introduction

Earth observation is a sequence of operations to collect, manage, process, analyze, and present Earth system-related physical, chemical, and biological data using remote sensing or other measuring methods. It has comprehensive applications and wide opportunities for the management of natural resources and tracking of the environment. With the advances in modern sensor technologies, various platforms, such as satellites, planes, and vehicles, have been employed as sensor carriers to gather various observation data, generating a wide range of data types and formats. On the other hand, the improved data resolution in time, space, and spectrum leads to a tremendous growth in EO data size and volume. For end customers, the information behind this data is more important. It is estimated, however, that some of the data were never accessed and processed, resulting in a waste of funds and incomplete information. Hence, how to organize, store, and manage large volumes of EO data and dig out available information from the data requires immediate attention.

Since most of the EO data, such as remote sensing images, are essentially multidimensional arrays in terms of data structure, it naturally follows to store and manage EO data in an array database, which is specially designed to manipulate data based on semantics of arrays. In addition, array databases offer good extensibility, flexibility, and efficient in-situ data processing capability.

1.1.1 Geospatial Big Data

Big Data is “extensive datasets — primarily in the characteristics of volume, variety, velocity, and/or variability — that require a scalable technology for efficient storage, manipulation, management, and analysis.”

Geospatial data has been Big Data for decades. New tools and technologies are now available to deal with Big Geo Data analytics and visualization. Geospatial data or information that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans, and more. Spatial data is usually stored as coordinates and topology, and is data that can be mapped.

The exponential growth of science and technology began in the 21st century, resulting in an explosion of information. Data had elements related to quantity, speed, variability, value, valence, and virtualization. Until recently, most of the information had never been evaluated and was discarded most of the moment. There was no technique to make full use of this enormous information. This situation came to be

known as “Big Data Problem”. Big Data issue not only entered the door of IT businesses but also reached administration, economy, sustainability, health care, research and growth, as well as all the sectors running on the machine globe as outlined by CL. Philip Chen's 2014 article. This issue also gave multiple organizations the chance to solve "Big Data issue" in their own manner; like Google solved it using GFS, social media giant Facebook used their own Big Data approach and so did the WalMart. Most of initial Big Data problem were not in Geo-Spatial domain.

However big data problem did touch ISRO’s Center for the analysis of Earth Observation data.

1.1.2 Array Database:

Array database management systems (DBMSs) provide database services specifically for arrays (also known as raster data), i.e. homogeneous data item collections (often known as pixels, voxels, etc.), sitting on a regular grid of one, two or more dimensions. Sensor, simulation, picture, or statistics information are often represented by arrays. Such arrays tend to be Big Data, often ranging from single objects to Terabyte and quickly to Petabyte sizes; for instance, today's earth and space observation archives typically expand by Terabytes a day.

Array databases on huge multi-dimensional arrays provide versatile, scalable services consists of storage management and multi-dimensional array processing features that forms a core science and engineering data structure. They were specifically intended to fill the gaps in the relational globe when dealing with large binary datasets of structured data and have gained traction in recent years in both science societies and industrial sectors such as agriculture, mineral resource exploitation, etc. 1-D sensor data, 2-D satellite and medical imagery, 3-D image timeseries, 4-D climate models are all at the core of virtually all science and engineering domains.

The significant increase in scientific data that occurred in the past decade – such as NASA’s archive growth from some hundred Terabytes in 2000 to 32 Petabytes of climate observation data – marked a change in the workflow of researchers and programmers. Early approaches consisted mainly of retrieving a number of files from an FTP server, followed by manual filtering and extracting, and then either running a batch of computation processes on the user’s local workstation, or tediously writing and optimizing sophisticated single-use-case software designed to run on expensive supercomputing infrastructures. This is not feasible any more when dealing with Petabytes of data which need to be stored, filtered and processed beforehand. When data providers discovered this they started providing custom tools themselves, often leading to silo solutions which turn out to erode over time and make maintenance and evolution hard if not impossible. An alternative finding attention only recently are database-centric approaches, as these have shown significant potential; meantime, we find both small institutions and large data-centers using modern database architectures for massive spatio-temporal data sets.

Arrays - also called "raster data" or "gridded data" or, more recently, "datacubes" - constitute an abstraction that appears in virtually all areas of science and engineering, and even beyond:

Table 1.1 Data Cube and V's of Big Data

Volume	Variety	Velocity	Veracity	Value
Store surface reflectance, bands and other products in Data Cube environment.	Sensor specific measurements into ARD.	Cube environments facilitate automated workflows(reduce manual work).	Create, manage and store all EO metadata using adopted standards.	Data extraction according to the need of analysis.

1.1.3 Raster data:

A raster consists of a matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information, such as temperature. Rasters are digital aerial photographs, imagery from satellites, digital pictures, or even scanned maps.

Data stored in a raster format represents real-world phenomena:

- Thematic data (also known as discrete) represents features such as land-use or soils data.
- Continuous data represents phenomena such as temperature, elevation, or spectral data such as satellite images and aerial photographs.
- Pictures include scanned maps or drawings and building photographs

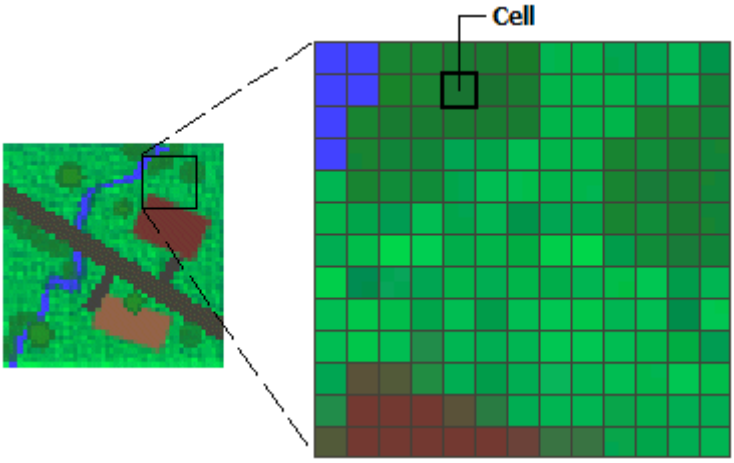


Fig 1.1: Raster data

While the structure of raster data is simple, it is exceptionally useful for a wide range of applications. Within a GIS, the uses of raster data fall under four main categories:

Rasters as basemaps:

A common use of raster data in a GIS is as a background display for other feature layers. Three main sources of raster basemaps are orthophotos from aerial photography, satellite imagery, and scanned maps.

Fig 1.2: Raster used as a basemap for road data

Rasters as surface maps:

Rasters are well suited to represent data that continually changes across a (surface) landscape. The most common application of surface maps are elevation values measured from the earth's surface, but other values, such as rainfall, temperature,

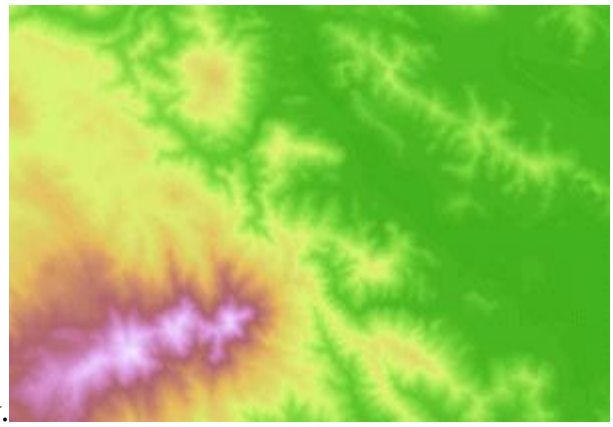
concentration, and population density, can also define surfaces that can be analyzed. spatially. 

Fig 1.3: Raster displays elevation using green to show lower elevation and red, pink, and white cells to show higher elevations.

Rasters as thematic maps:

Rasters representing thematic data can be derived from analyzing other data. A common analysis application is classifying a satellite image by land-cover categories.

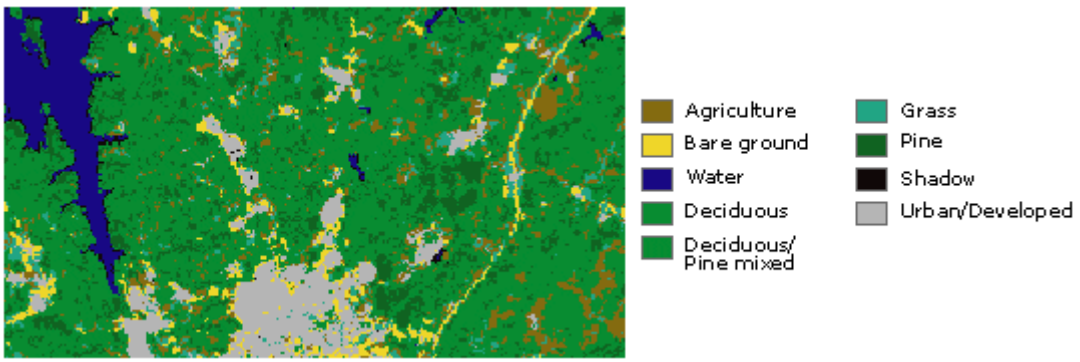


Fig 1.4: Classified raster dataset showing land

Each cell (also known as a pixel) has a value in raster datasets. The cell values represent the phenomenon represented as a category, magnitude, height, or spectral value by the raster dataset. A class of land-use such as grassland, forest, or highway could be the category. A magnitude could be gravity, noise pollution, or precipitation percentage. Height (distance) may depict ground elevation above mean sea level, which may be used to derive characteristics of slope, appearance, and watershed. Spectral values are used to depict light reflection and color in satellite imagery and aerial photography.

1.1.4 Satellites

ISRO Satellites:

RESOURCESAT-2 is a follow on mission to RESOURCESAT-1 and the eighteenth Remote Sensing satellite built by ISRO. RESOURCESAT-2 is intended to continue the remote sensing data services to global users provided by RESOURCESAT-1, and to provide data with enhanced multispectral and spatial coverage as well. Varieties of instruments have been flown onboard these satellites to provide necessary data in a diversified spatial, spectral and temporal resolutions to cater to different user requirements in the country and for global usage. The data from these satellites are used for several applications covering agriculture, water resources, urban planning, rural development, mineral prospecting, environment, forestry, ocean resources and disaster management.

NASA Satellites:

Landsat 7 is the seventh satellite of the Landsat program. Launched on April 15, 1999, Landsat 7's primary goal is to refresh the global archive of satellite photos, providing up-to-date and cloud-free images. The Landsat Program is managed and operated by the USGS, and data from Landsat 7 is collected and distributed by the USGS. The NASA World Wind project allows 3D images from Landsat 7 and other sources to be freely navigated and viewed from any angle. The satellite's companion, Earth Observing-1, trailed by one minute and followed the same orbital characteristics, but in 2011 its fuel was depleted and EO-1's orbit began to degrade. Landsat 7 was built by Lockheed Martin Space Systems Company.

In 2016, NASA announced plans to attempt the first ever refueling of a live satellite by refueling Landsat 7 in 2020.

Landsat 5 was launched by NASA from Vandenberg Air Force Base on March 1, 1984. The Landsat 5 satellite orbited the the Earth in a sun-synchronous, near-polar orbit, at an altitude of 705 km (438 mi), inclined at 98.2 degrees, and circled the Earth every 99 minutes. The satellite had a 16-day repeat cycle with an equatorial crossing time: 9:45 a.m. +/- 15 minutes. Landsat 5 data were acquired on the Worldwide Reference System-2 (WRS-2) path/row system, with swath overlap (or sidelap) varying from 7 percent at the Equator to a maximum of approximately 85 percent at extreme latitudes.

1.2 Problem Definition

Every year, fresh generations of Earth Observation satellites are generating petabytes of information with such extensive worldwide coverage that information shortages are no longer a limiting factor for many apps. Extensive R&D activity has provided fresh information apps that offer significant potential to greatly affect major environmental, economic, and social problems, including at local, regional, and global levels. Such

applications emphasize EO's importance, although the task is to provide the right links between information, apps and customers. Even today, much archived EO satellite data is underutilized despite modern computing and analysis infrastructures.

Multi-level analysis presents a challenge, i.e. it is necessary to find, recognize and evaluate geographic items at various scales. For example: Urban planners are interested in up-to-date land coverage and land use data on urban objects at multiple spatial scales (1:100,000 to 1:5,000) and temporal scales Geologists and geophysicists are interested in precise mapping of landslides, offering appropriate data on future environmental and/or human hazards and a clearer understanding of the landslide structure through anal analysis.

Multi-temporal analysis is another challenge. Two kinds of multi temporal modifications: 1. Long-term shifts (e.g., town development) 2. Cyclical shifts (e.g. farming). Some phenomena are distorted temporarily. Every state of urbanization (barren soil, fresh house, ancient house) can be distorted temporarily. Agronomy Shifted harvesting / collecting from parcel to parcel. Speed difference in Ripening / Maturation.

Big Data Challenges such as High frequency of observations, High frequency of observations impacts methodologies i.e. to update databases or background knowledge to extract geographic object evolutions, high frequency associated to big volume makes that unrealistic to be able to process all the data for each new observation. Major problem is how to extract information from image time series with high frequency.

It is hard for developed economies to address this challenge and even more challenging for developing nations interested in using satellite information from EO. It is merely not technically viable or financially accessible to consider traditional techniques of local processing and information distribution (e.g. online scene-based file download) to tackle this "scaling" challenge in many countries, as information size and complexities in preparing, handling, storage, and evaluation remain important. barriers.

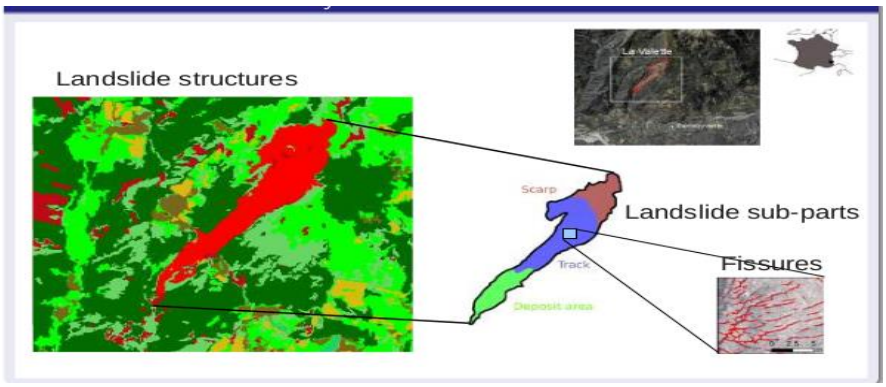


Fig 1.5: Multi-level analysis

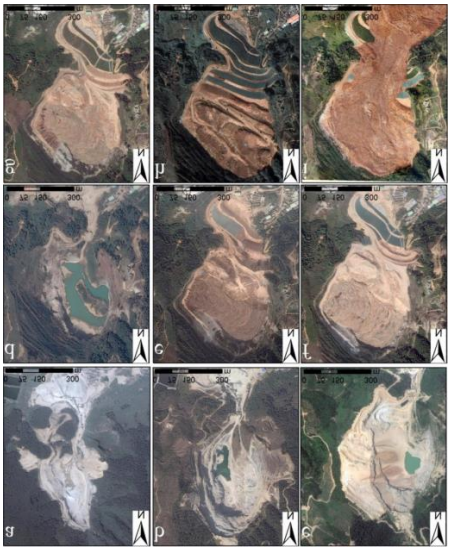


Fig 1.6: Multi Temporal satellite imagery

1.3 Aim

The aim of this project is to provide an efficient way of managing Earth Observation big data (Satellite images) and to improve the efficiency of time series data analysis using automated algorithms.

1.4 Objectives

To provide a flexible, generic and broad solution is required to analysis remote sensing time series data, avoiding the burden of development and adaptation according to the scientific needs, the objectives of our project are:

1. Installation and Configuration of Array Database (Open Data Cube).
2. Development of ingestion module and creation of Data Cube.
3. Implementation of algorithms for various indices such as NDVI, NDWI etc.
4. Development of web application for time series geospatial data analysis.

Open Data Cube stores multidimensional data in array of cubes, which provides better management of satellite data. Web application will provide better analysis of geospatial data, which will fulfill the requirement of scientific needs.

CHAPTER-2

LITERATURE STUDY

2.1 Introduction

Over time, geographic data quantity and accessibility has risen exponentially. This creates a fresh challenge, as distinct sources of information–authoritative data (e.g. sensor data) and voluntary geographic information–need to be integrated to enhance situational consciousness and support decision-making.

2.2 Implemented Methods

Satellite imagery has long been spread on big tape rolls. Less long ago, in 2011, as part of the Landsat Archive Unlock initiative, Geoscience Australia worked with several other organizations to copy the Landsat information they had stored on tapes and elsewhere on spinning disks at the National Computational Infrastructure. Some time after this, the Australian Geoscience Data Cube (AGDC) was developed, a Landsat specific tool able to enhance access to these Landsat archives.

More recently, AGDC was rewritten as AGDCv2. This newer version had a number of goals in mind, including supporting arbitrary coordinate reference systems and file formats, and keeping track of processing provenance. In 2017, AGDCv2 was renamed the Open Data Cube and governance structures were set up to ensure that the project could have continuing long term support.

Traditional storage for EO data uses various kinds of files, such as Network Common Data Form (NetCDF) for atmospheric and hydrological sciences, GeoTIFF, and Hierarchical Data Format (HDF) for remote sensing images. These speciallydesigned data formats work quite well when the amount of data is not very large. However, issues start to arise when data volumes increase gradually. The most obvious problem is that it is not easy to retrieve and query the information needed. To solve this problem, there have been some efforts to provide a more productive environment for EO data storage and processing by leveraging the state-of-the-art multidimensional array databases and High-Performance Computing (HPC) technologies. [1]

With wider interest in, and the increasing volume, variety and velocity of, EO data, new challenges arise. There is a need to find the best way to store and manage data, and to make it accessible for end-users without the need for downloading. Need to produce useful information systematically from the content of an image, or even a long time series of images. Need to find out the way to combine EO data from different sensors, or to combine them with non-EO data. Arguably, in most analyses only a small portion of the available images are currently being used. There are

several possible reasons for this. The prevalent requirement is still for data to be downloaded prior to analysis, partly because local workflows cannot be easily translated into Web-based workflows. Other reasons are the tools and computation capabilities that are simply not powerful enough yet, and the lack of interoperability and transferability of approaches. [2]

An array database is designed and implemented as a common database service offering flexible and scalable storage and retrieval on large volumes of multidimensional array data, such as sensor, image, simulation or statistics data. It has attracted extensive attention from academic and industry data scientists. In the geoscience domain, for example, the National Aeronautics and Space Administration (NASA) launched a project—EarthDB to manage tons of Moderate Resolution Imaging Spectroradiometer (MODIS) Level 1B calibrated and georeferenced data by employing an open-source array database named SciDB. The EarthServer is a European Union-funded project to meet the EO Big Data challenges. With all the data stored in Rasdaman array database, it provides a flexible and interoperable processing and analytic service based on Open Geospatial Consortium(OGC) standards. The Australian GeoScience Data Cube (AGDC) is another project that establishes a comprehensive EO Big Data storage and processing framework by employing multidimensional array-based storage and HPC technologies. These are excellent works contributing to the geoscience community.

New Processing method i.e. Map-Reduce Framework ("MapReduce is a programming model and an associated implementation for processing and generating large data sets"), "a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key". Programs written in this model will automatically parallelized and can be run on large cluster of desktop machines without high end configuration. Basically, if you want to run Map-Reduce function, the real world task must be converted into Map-Reduce function type, else it would be tough to implement this model. Fortunately most of the real world task can be converted into Map-Reduce function type. [3]

If Map-Reduce framework is to be utilized then Geo-Spatial big data problem must be converted into 2 set of functions: one doing Mapping and other doing reducing. Once this is done then system will take care of run-time data partitioning, scheduling of programs to be executed across the cluster, fault tolerance and inter-machine communication.

This framework is data-centric structure in which there was no spatial orientation. What Hadoop offers is a distributed file system and a data transformation framework.

Hadoop became Yahoo's Map-Reduce framework software application. This project later turns into an open source project. Hadoop's opening as an open source converted the entire information industry, which was clueless when faced with the big data issue. But the solution of some confusions was still Map-Reduce's execution architecture with regard to spatial data and the capablity of Hadoop's Map-Reduce implementation system with regard to spatial data.

The two main issue of managing spatial partitioning is: one, it could lead to issue of load-balancing and it could slow down the system and the other, it could be incorrect responses. Hadoop-GIS has been created to provide a extremely scalable, cost-effective and effective spatial querying scheme. The querying was split into tiny tasks and processes in order to understand this scheme. The information was divided into tiles or blocks in space and then processed in parallel. The querying would be done on these tiles. The resulted data would be then joined maintaining semantics. However, in managing spatial data, Hadoop is ill-equipped as it treats spatial and non-spatial information in the same way. It only supports a uniform grid index, which only applies in cases where the distribution of data is uniform. Hadoop-GIS has been designed to overcome constraints; an extent that is spatially conscious has been constructed and called its Spatial Hadoop.

The growth of any science depends on several conditions being met. First, there must be good-quality data available. Second, there must be well-formulated hypotheses that can be formalized mathematically, so that they can be subject to empirical testing. Third, there must be a rigorous methodology that enables the analyst to draw valid inferences and conclusions from the data in relation to the questions asked. This includes the ability to formulate models that can then be used to test hypotheses on parameters of interest. The final condition is the availability of a technology that permits research to be undertaken practically and to acceptable standards of precision.

2.3 Conclusion

Based on the work of pioneer contributors, it is thought that three elements of EO Big Data storage should be taken into account: (1) method of storing big quantities of data despite their varied formats and distinct observation topics; (2) method of querying and extracting helpful information from huge data archives and helping to figure out precisely what consumers need; (3) managing and maintaining the data. The following three guidance strategies are identified accordingly. Firstly, common characteristics of EO data should be summarized and concluded to choose a general and consistent storage and management platform. Secondly, easy-to-use query languages or utility tools should be developed to accurately locate the data required. Finally, the storage system for EO Big Data must be as flexible and scalable as possible.

CHAPTER-3

TOOLS AND METHODOLOGY

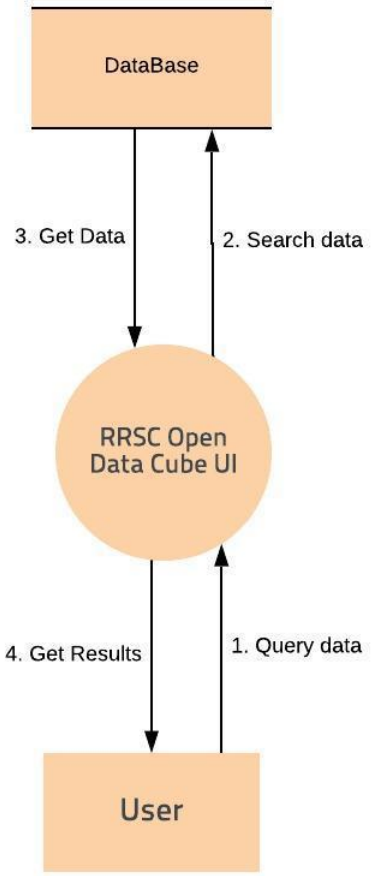


Fig 3.1: DFD Level 0

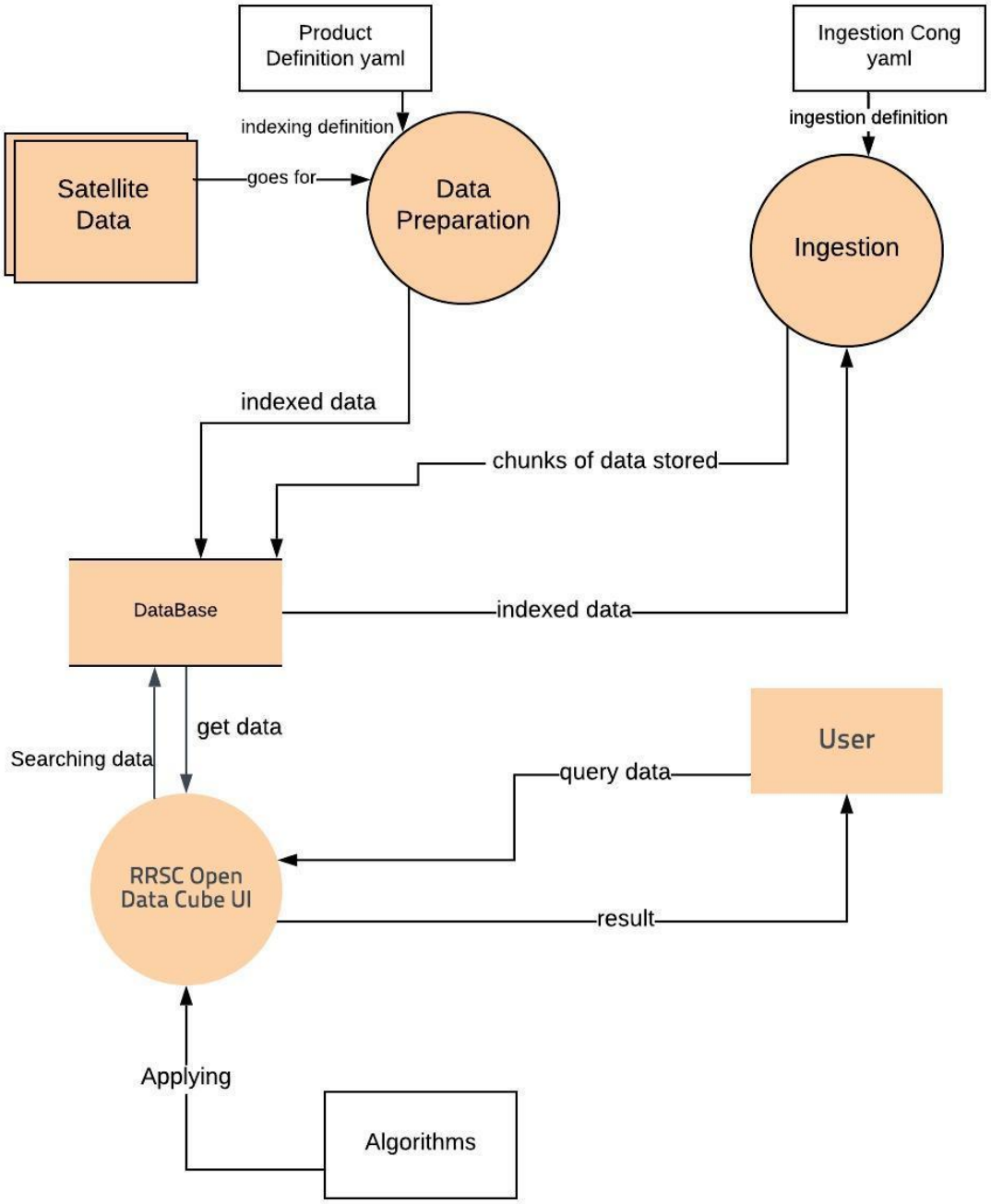


Fig 3.2: DFD Level 1

3.1 Tools

3.1.1 System Requirements

- OS: Ubuntu 18.04 LTS
- Memory: 8GiB
- Local Storage: 50GiB
- Python Version: Python 3

3.1.2 Software Requirements

3.1.2.1 Open Data Cube

Time series multidimensional stack of spatially aligned pixels ready for analysis. It is designed to-

- Catalogue large amount of Earth Observation data.
- Provide python API for high performance querying & data access.
- Allow scalable continent scaling processing of stored data.

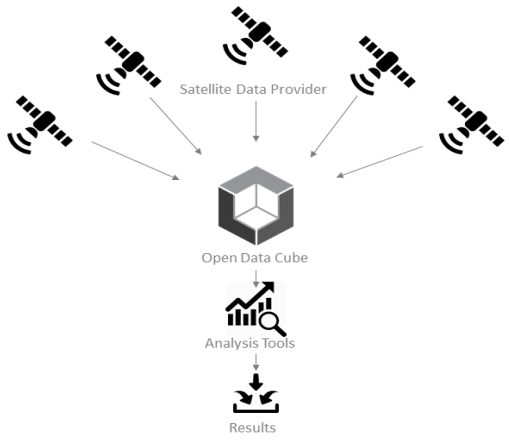


Fig 3.3: Open Data Cube Overview

The Open Data Cube provides an integrated gridded data analysis environment for decades of analysis ready earth observation satellite and related data from multiple satellite and other acquisition systems.

3.1.2.2 PostgreSQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build

fault tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database.

3.1.2.3 GDAL

The Geospatial Data Abstraction Library (GDAL) is a computer software library for reading and writing raster and vector geospatial data formats. GDAL/OGR is considered a major free software project for its "extensive capabilities of data exchange" and also in the commercial GIS community due to its widespread use and comprehensive set of functionalities.

As a library, it presents a single abstract data model to the calling application for all supported formats. It may also be built with a variety of useful command line interfaceutilities for data translation and processing. Projections and transformations are supported by the PROJ.4 library.

3.1.2.4 Web Application Tools

Front-end-

- HTML
- Java Script
- CSS
- Python

Back-end-

- Postgresql
- Apache WebServer

3.2 Datasets

There are four types of datasets have been taken to store and process.

1. Landsat_7:

Landsat 7 Enhanced Thematic Mapper Plus (ETM+)	Bands	Wavelength (micrometers)	Resolution (meters)
	Band 1 - Blue	0.45-0.52	30
	Band 2 - Green	0.52-0.60	30
	Band 3 - Red	0.63-0.69	30
	Band 4 - Near Infrared (NIR)	0.77-0.90	30
	Band 5 - Shortwave Infrared (SWIR) 1	1.55-1.75	30
	Band 6 - Thermal	10.40-12.50	60 * (30)
	Band 7 - Shortwave Infrared (SWIR) 2	2.09-2.35	30
	Band 8 - Panchromatic	.52-.90	15

Fig 3.4: Landsat_7 bands

Downloaded an area of Maharashtra nearby Nagpur having 2 scenes of 2002 which is of 2 GBs (approx.) from USGS Earth Explorer.

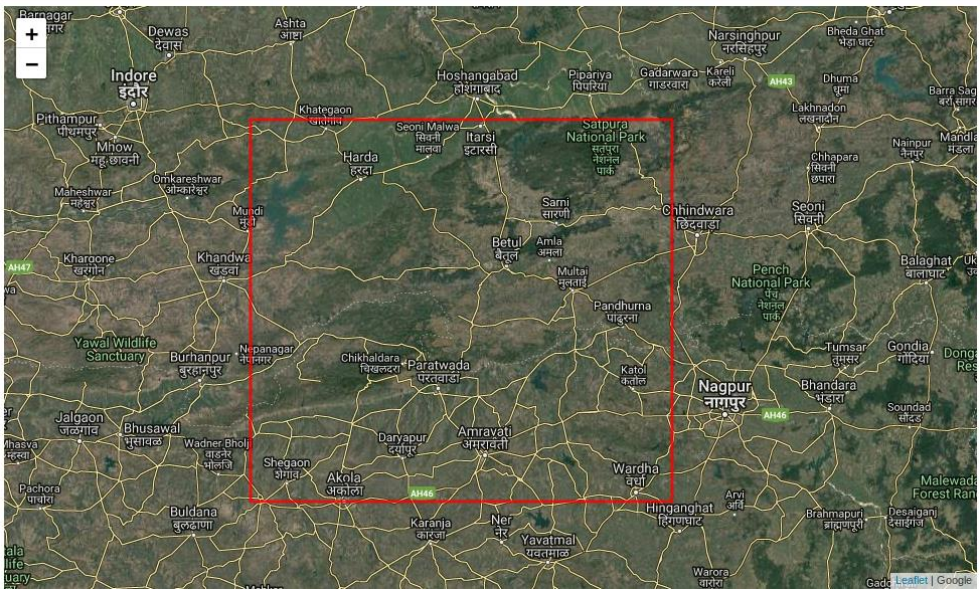


Fig 3.5: Scene Coverage Landsat_7

2. Resourcesat-2:

LISS-3 (Linear Imaging Self Scanning sensor-3) of ResourceSat-2		
Bands	Wavelength (µm)	Resolution (m)
-	-	-
-	-	-
B2 - Green	0.52 - 0.59	24
B3 - Red	0.62 - 0.68	24
B4 - NIR	0.77 - 0.86	24
B5 - SWIR	1.55 - 1.70	24
-	-	-
-	-	-

Fig 3.6: Resourcesat-2 bands

Downloaded an area inside Madhya Pradesh nearby Gwalior having 4 scenes from 2014-2016 of about 50 MBs from Bhuvan's site.

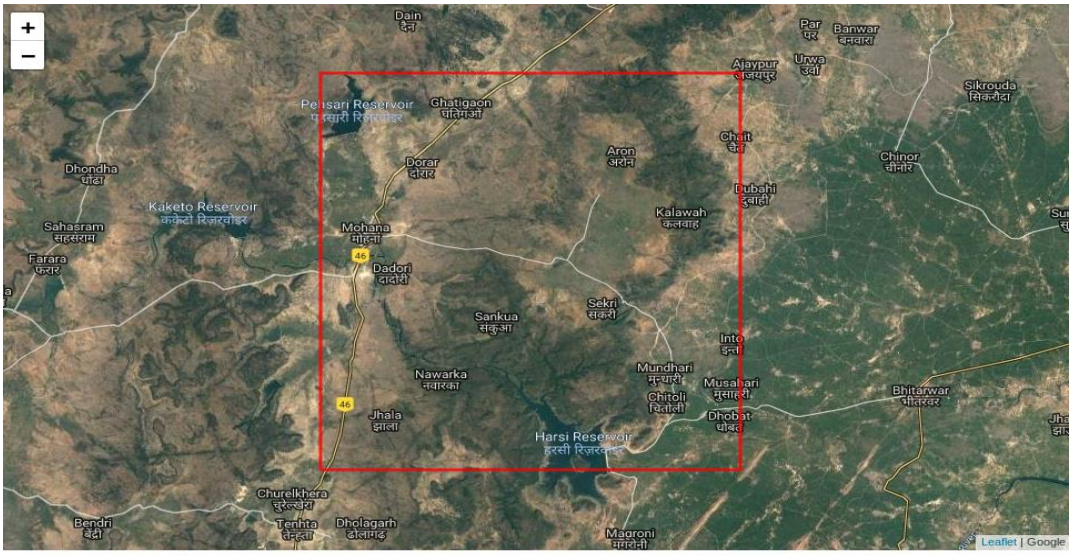


Fig 3.7: Scene Coverage Resourcesat-2

3. LANDSAT-5:

Landsat 4-5 Thematic Mapper (TM)	Bands	Wavelength (micrometers)	Resolution (meters)
	Band 1 - Blue	0.45-0.52	30
	Band 2 - Green	0.52-0.60	30
	Band 3 - Red	0.63-0.69	30
	Band 4 - Near Infrared (NIR)	0.76-0.90	30
	Band 5 - Shortwave Infrared (SWIR) 1	1.55-1.75	30
	Band 6 - Thermal	10.40-12.50	120* (30)
	Band 7 - Shortwave Infrared (SWIR) 2	2.08-2.35	30

Fig 3.8: Landsat_5 bands

33 scenes of Landsat_5 of an area nearby Nagpur (Maharashtra) of 39 GBs (approx.) has been downloaded from USGS Earth Explorer.

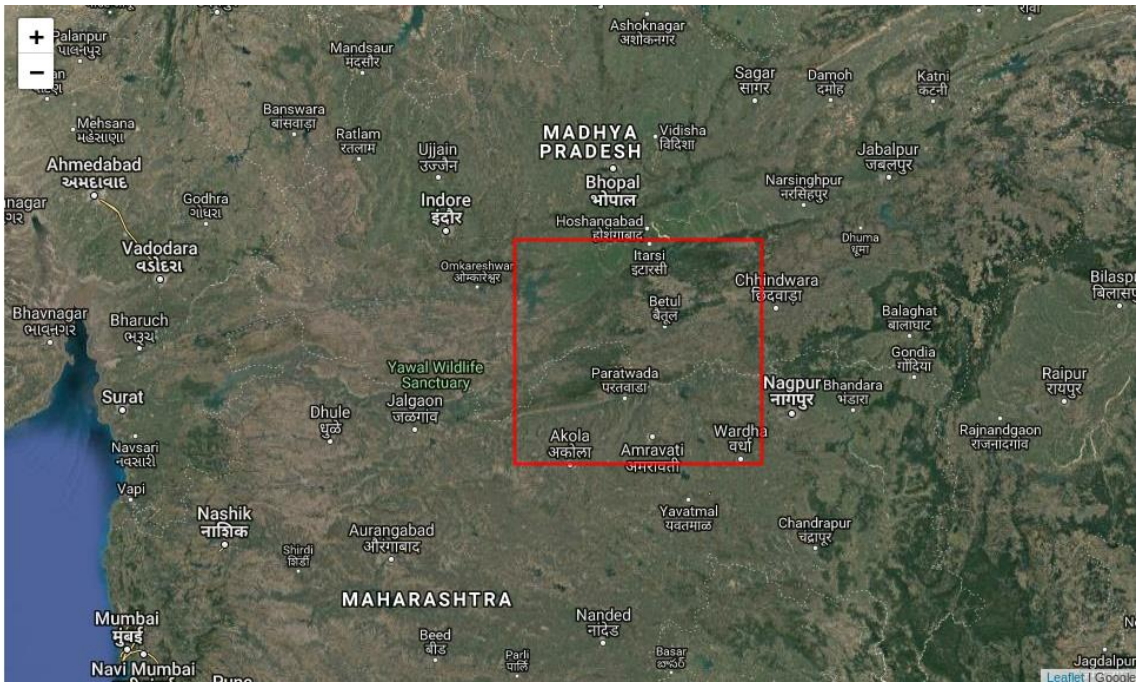


Fig 3.9: Scene Coverage Landsat_5

3.3 Methodology:

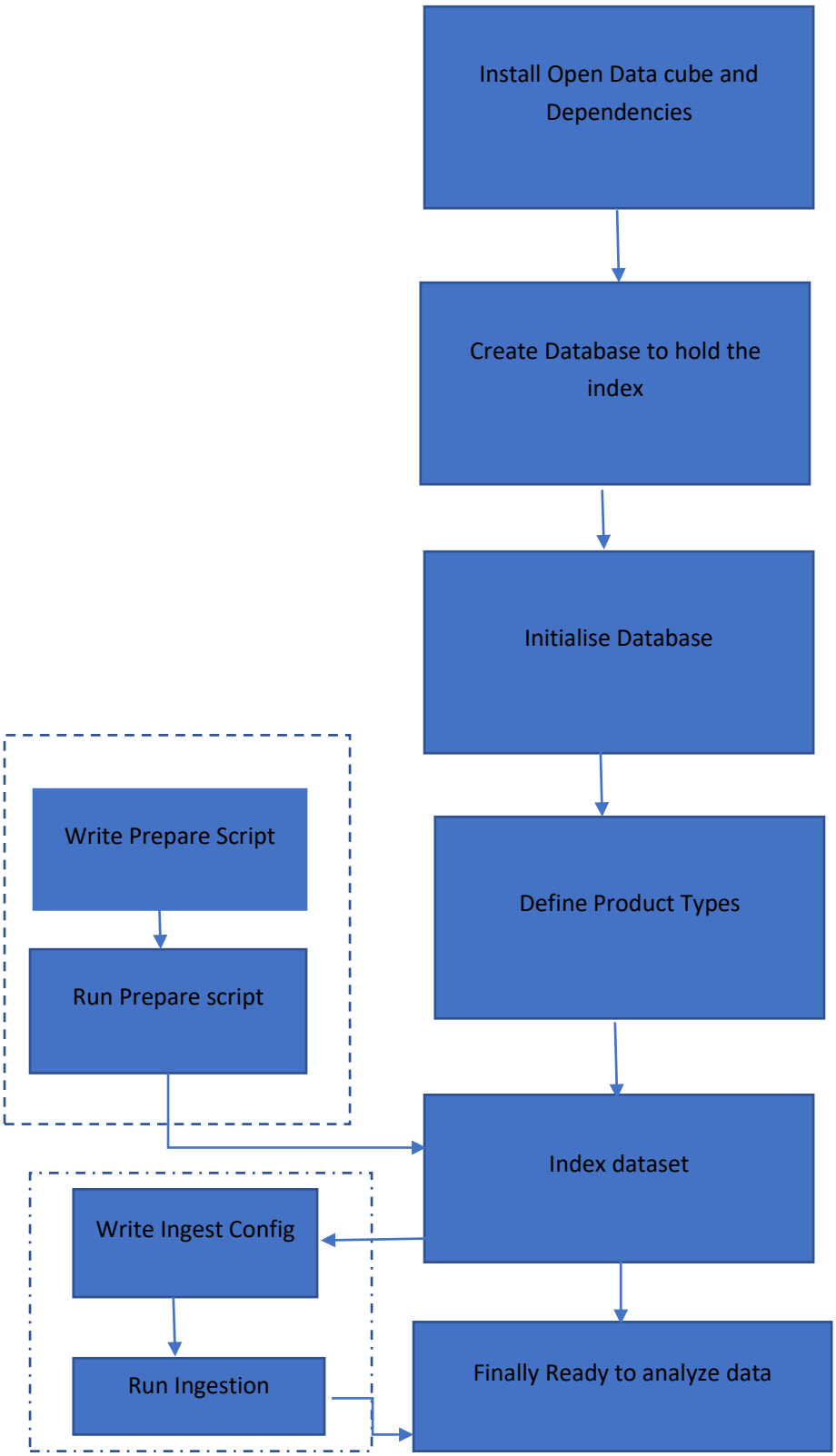


Fig 3.10: Open Data Cube Installation

Step 1: Installation of Data Cube, which consist of python and so many dependencies.

The Data Cube is a set of python code with dependencies including:

- Python 3.5+ (3.6 recommended)
- GDAL
- PostgreSQL database

Step 2: Setting up the Database and Environment Configuration

The database used by the Data Cube is Postgresql. The datacube Database schema can be created by system init tool. Environment configuration includes the development of configuration file which will link database to the datacube.

Step 3: Indexing of Data

It includes three steps to be followed:

- Product Definition File
- Add product Definition
- Metadata Generation
- Add Data-Indexing

Product Definition File: It is a file which contains all the details about the metadata set. In our case the metadata type is eo i.e. Earth Observation and also contains its fields such as product_type, instrument name, satellite name and its measurements means details of all the bands of the satellite image that has to be ingested.

For example, product definition file of LISS-III Resourcesat-2 which consist of only four bands:

```
name: l3_scene
description: Resourcesat-2,LISS-III,24m,India,ISRO,NRSC,Ortho
metadata_type: eo

metadata:
  format: {name: GeoTiff}
  instrument: {name: LISS-III}
  platform: {code: Resourcesat-2}
product_type: Path-Row

measurements:
- dtype: uint16
  name:  'band2'
  aliases: [green]
  nodata: 0
  units: 'reflectance'

- dtype: uint16
  name:  'band3'
```

```
    aliases: [red]
    nodata: 0
    units: 'reflectance'

  - dtype: uint16
    name:  'band4'
    aliases: [nir]
    nodata: 0
    units: 'reflectance'

  - dtype: uint16
    name:  'band5'
    aliases: [swir]
    nodata: 0
    units: 'reflectance'
```

Adding Product Definition: Product Definition file that was created has to be added into the DataCube database for future use with a single line command.

Metadata generation: To generate the metadata of dataset, it has to be downloaded and stored on local disk under the specific defined folder.
To generate the metadata there is a need of python script, which takes the xml file downloaded with the dataset and the specific folder name of the dataset.
In case of Resourcesat-2 there is no xml file downloaded which consist of details of all the bands in the scene, so first it has to be created with the help of the bands configuration as can be viewed in QGIS (a geospatial software to view and anlaysis the bands of the scene) or can get details of the scene through GDAL commands in the terminal.

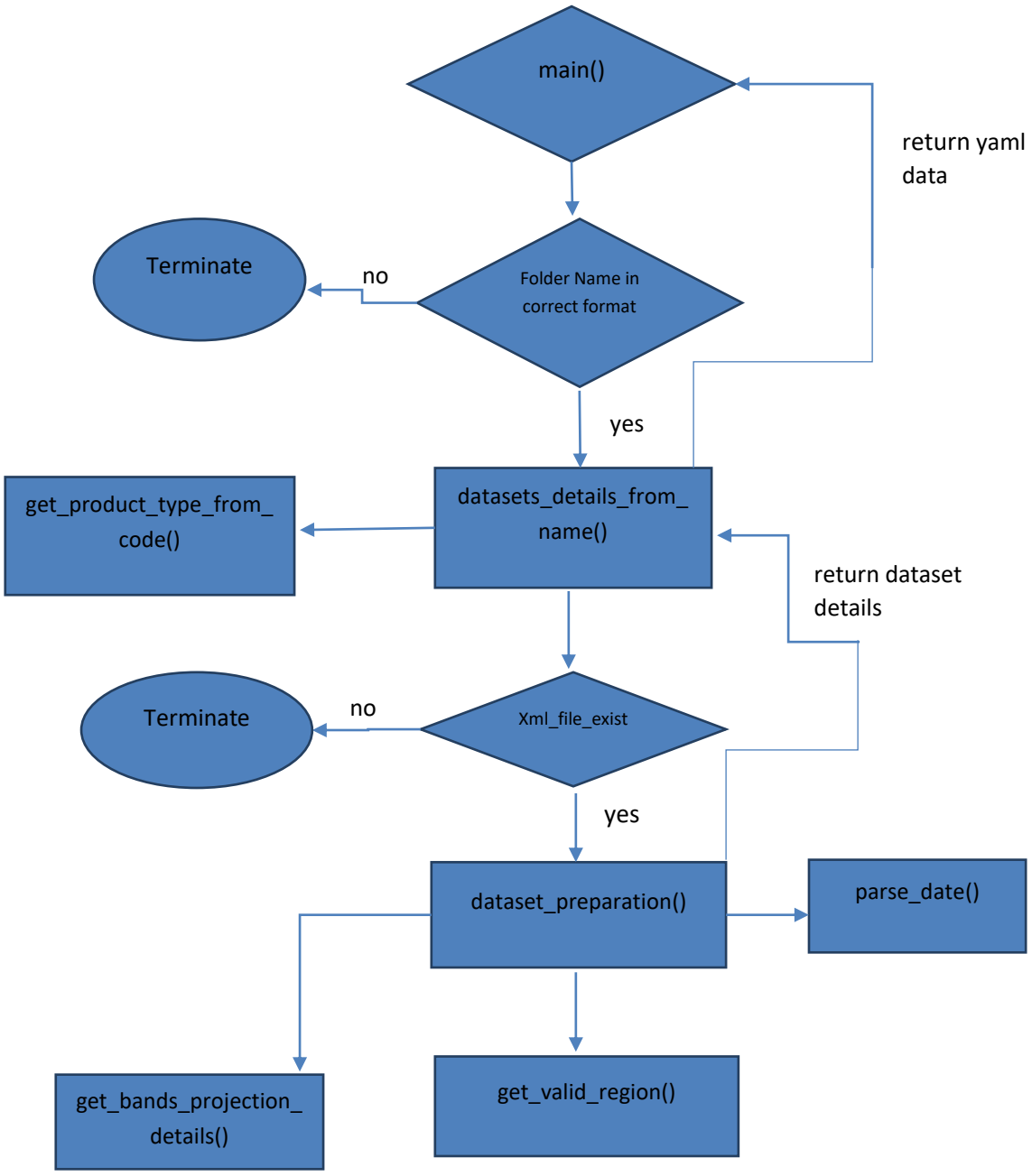


Fig 3.11: Python Script Flowchart

Generated YAML Metadata File: The generated metadata file is in YAML format and it consist of all bands details, product-type, satellite name and the instrument i.e. sensor name and the date on which scene has been taken.
Example: Metadata file of LISS-III Resourcesat-2

```
acquisition: {aos: '2016-01-07 23:59:48', los: '2016-01-08 00:00:12'}
creation_dt: 0016-01-08 00:00:00
extent:
center_dt: '2016-01-08 00:00:00'
coord:
ll: {lat: 25.7455, lon: 77.7455}
lr: {lat: 25.7455, lon: 78.0045}
ul: {lat: 26.0045, lon: 77.7455}
ur: {lat: 26.0045, lon: 78.0045}
from_dt: '2016-01-07 23:59:48'
to_dt: '2016-01-08 00:00:12'
format: {name: GeoTiff}
grid_spatial:
  projection:
geo_ref_points:
ll: {x: 77.7455, y: 25.7455}
lr: {x: 78.0045, y: 25.7455}
     ul: {x: 77.7455, y: 26.0045}
ur: {x: 78.0045, y: 26.0045}
spatial_reference: GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],P
RIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,A
UTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]
valid_data:
     coordinates:
     - - [78.0045, 26.0045]
       - [77.7455, 26.0045]
       - [77.7455, 25.7455]
       - [78.0045, 25.7455]
       - [78.0045, 26.0045]
     type: Polygon
id: 975d794b-cbd2-4da2-ab48-e98cdf566417
image:
  bands:
    band2: {path: L3-NG43R13-097-053-08Jan16-band2.tif}
    band3: {path: L3-NG43R13-097-053-08Jan16-band3.tif}
    band4: {path: L3-NG43R13-097-053-08Jan16-band4.tif}
    band5: {path: L3-NG43R13-097-053-08Jan16-band5.tif}
satellite_ref_point_end: {x: 97, y: 53}
satellite_ref_point_start: {x: 97, y: 53}
instrument: {name: LISS-III}
lineage:
source_datasets: {}
platform: {code: Resourcesat-2}
product_type: Path-Row
```


Add Data-Indexing: The last step of indexing is to index the data in the database and associate with the product definition. It will index only if the metadata matches with any specific product definition file. Indexing is done through only a single datacube command “datacube dataset add”. Indexing the dataset in the database creates an absolute reference to the path on disk.

Step 4: Testing of Dataset:

Testing of Data Cube API access for the indexed dataset. The script has generated .yaml file for the GeoTiffdataset and on qurying the datset it will display the data in xArray dataset. For the sake of simplicity all the testing has been done through python console. Then, all normal numpy and xArray functions can be applied to Data Cube data.

Step 5: Ingestion of Datasets:

Ingestion is the process of transforming original datasets into more accessible format that can be used by the Data Cube. Like the previous steps, ingestion relies on the use of a .yaml configuration file that specifies all of the input and output details for the data. The ingestion configuration file contains all of the information required for a product definition - it uses the information to create a new product definition and index it in the Data Cube. Next, indexed source datasets that fit the criteria are identified, tiled, reprojected, and resampled (if required) and written to disk as NetCDF storage units. The required metadata is generated and the newly created datasets are indexed (added) to the Data Cube.

The ingestion configuration file is essentially defining a transformation between the source and the output data - we can define attributes such as resolution, projection, tile sizes, bounds, and measurements, along with file type and other metadata in a configuration file. When we run the ingestion process, the Data Cube determines what data fits the input data attributes by product type and bounds, applies the defined transformation, and saves the result to disk and in the database. This process is generally done when the data is in a file format not optimized for random access such as GeoTiff - NetCDF is the preferred file type.

Example of LISS-III Resourcesat-2 ingestion config .yaml file:

```
source_type: l3_scene
output_type: l3_general
description: Resourcesat 2 scene processed using LISS III.EPSG:4326 resolution
location:
'/datacube/ingested_data'file_path_template:'L3_General5/General/L3_4326_{tile_ind
ex[0]}_{tile_index[1]}_{start_time}.nc'

global_attributes:
storage:
driver: NetCDF CF
crs: EPSG:4326
tile_size:
 longitude: 0.943231048326
latitude:  0.943231048326
resolution:
        longitude: 0.000269494585236
        latitude: -0.000269494585236
```

```
  chunking:
      longitude: 150
      latitude: 150
      time: 1
dimension_order: ['time', 'latitude', 'longitude']
measurements:
     - name: green
dtype: uint16
nodata: 0
resampling_method: nearest
src_varname: 'band2'
zlib: True
attrs:
long_name: "Surface Reflectance Band 2 Green"
          alias: "green"
 - name: red
dtype: uint16
nodata: 0
resampling_method: nearest
src_varname: 'band3'
zlib: True
attrs:
long_name: "Surface Reflectance Band3 Red"
          alias: "red"
 - name: swir
dtype: uint16
nodata: 0
resampling_method: nearest
src_varname: 'band4'
zlib: True
attrs:
long_name: "Surface Reflectance Band 4 SWIR"
          alias: "swir"
```

Use Case Diagram

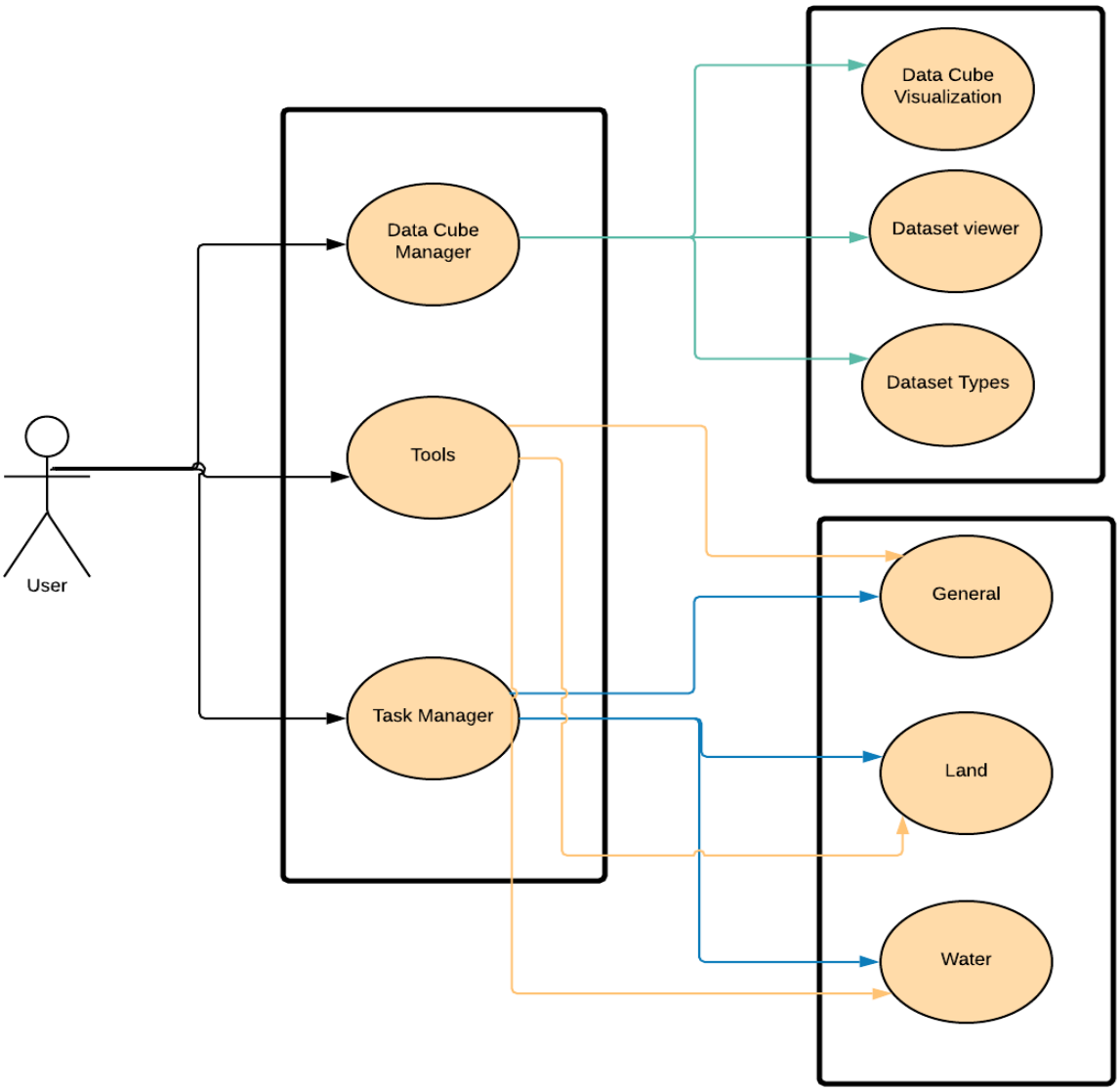


Fig 3.12: Data Cube UI: Top Level Use Case

CHAPTER 4

ALGORITHMS

4.1 NDVI Calculation

Normalized Difference Vegetation Index (NDVI) quantifies vegetation by measuring the difference between near-infrared (which vegetation strongly reflects) and red light (which vegetation absorbs).

NDVI always ranges from -1 to +1. But there isn't a distinct boundary for each type of land cover.

For example, when you have negative values, it's highly likely that it's water. On the other hand, if you have a NDVI value close to +1, there's a high possibility that it's dense green leaves. But when NDVI is close to zero, there isn't green leaves and it could even be an urbanized area.

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)}$$

Healthy vegetation (chlorophyll) reflects more near-infrared (NIR) and green light compared to other wavelengths. But it absorbs more red and blue light. This is why our eyes see vegetation as the color green. If you could see near-infrared, then it would be strong for vegetation too. Satellite sensors like Landsat and Sentinel-2 both have the necessary bands with NIR and red.

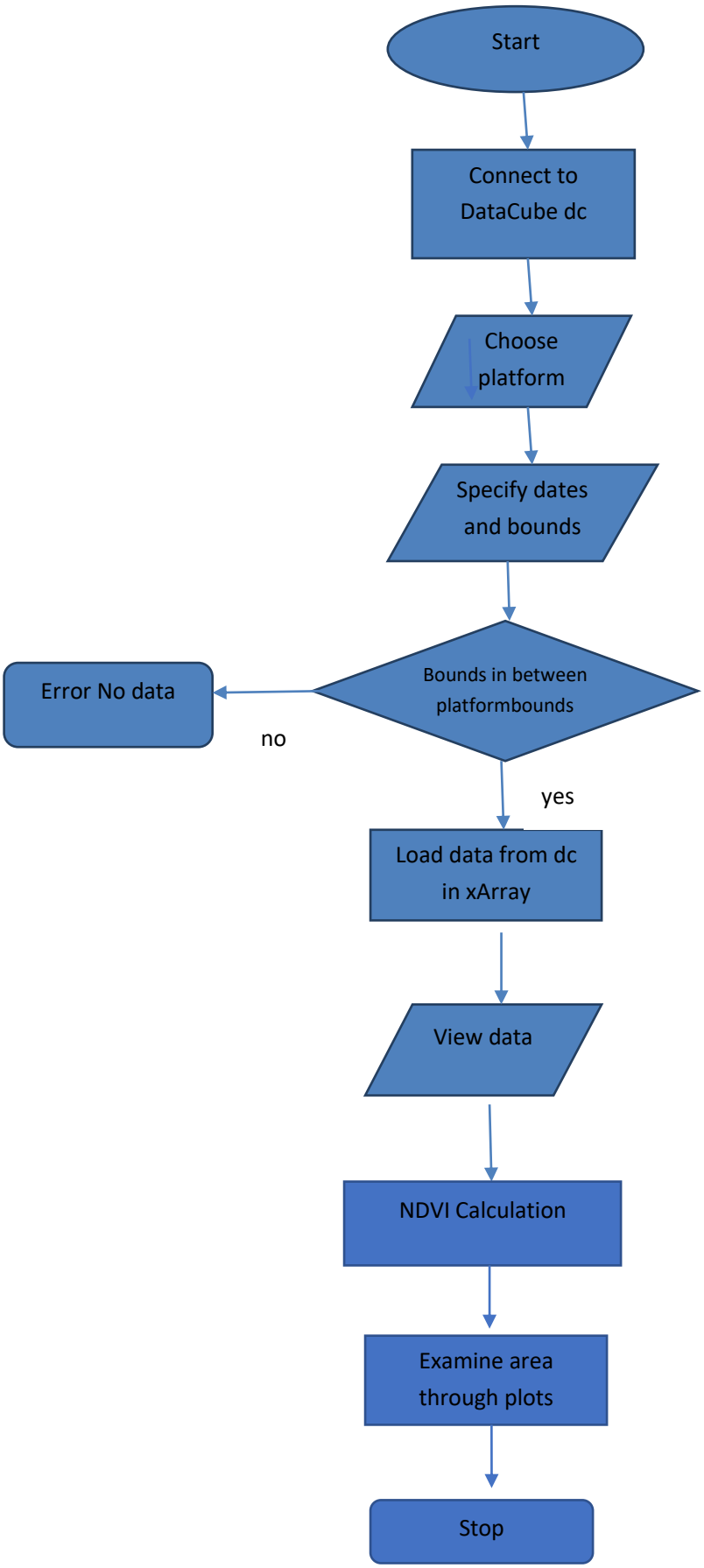


Fig 4.1: NDVI Calculation Algorithm Flowchart

First of all there is a need to connect to the database datacube, here we named it as dc. The database consist of all the indexed and ingested datasets. In the next step, all the ingested platforms will be displayed, there is an need to select the specific platform on which algorithm has to be processed. Ingested data contains bounds i.e. latitude and longitude of the area, time attribute shows the acquisition date of the satellite scene. As the data might contains GBs of data, so if the algorithm will perform on full data it will result in "memory error", so as a precaution we need to choose our area of interest i.e. specify latitude and longitude and can specify time bounds too. It might be possible that the selected time bounds does not lie between bounds of the ingested data, so for this there is condition to check that, if scene lies between the bounds then its information will be displayed as xArray dataset. The dataset can also be viewed on map through "display map" function by passing lat and long bounds as parameters.

If the dataset contain nir and red band then NDVI algorithm will work. In the next step NDVI calculation will performed. The result of NDVI will also be in the form of xArray dataset, which can be viewed through matplotlib utils.

Result:

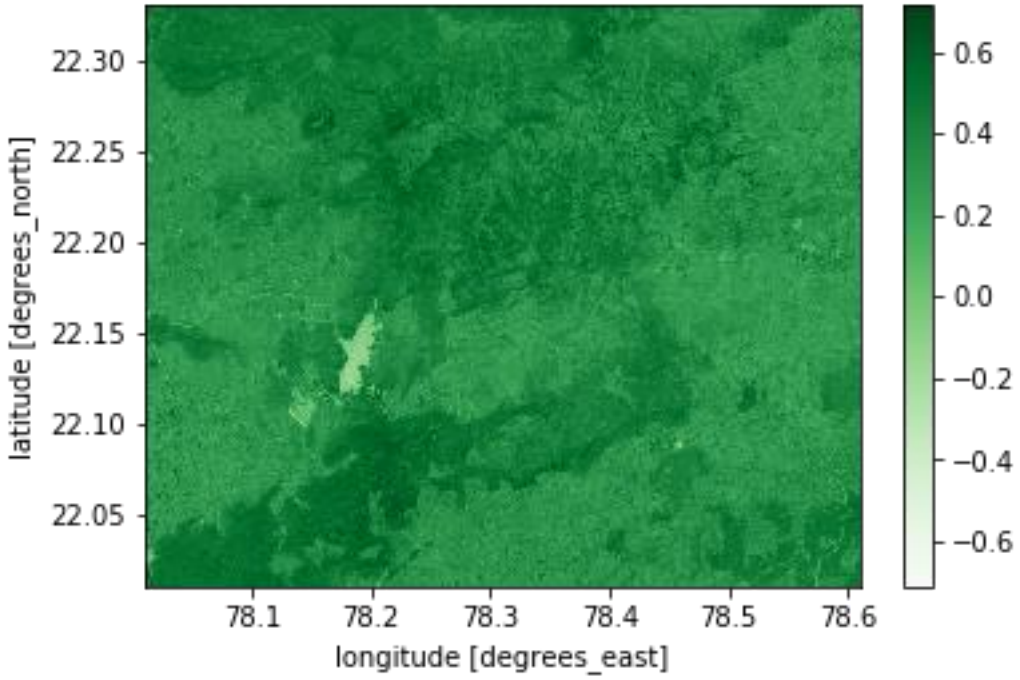
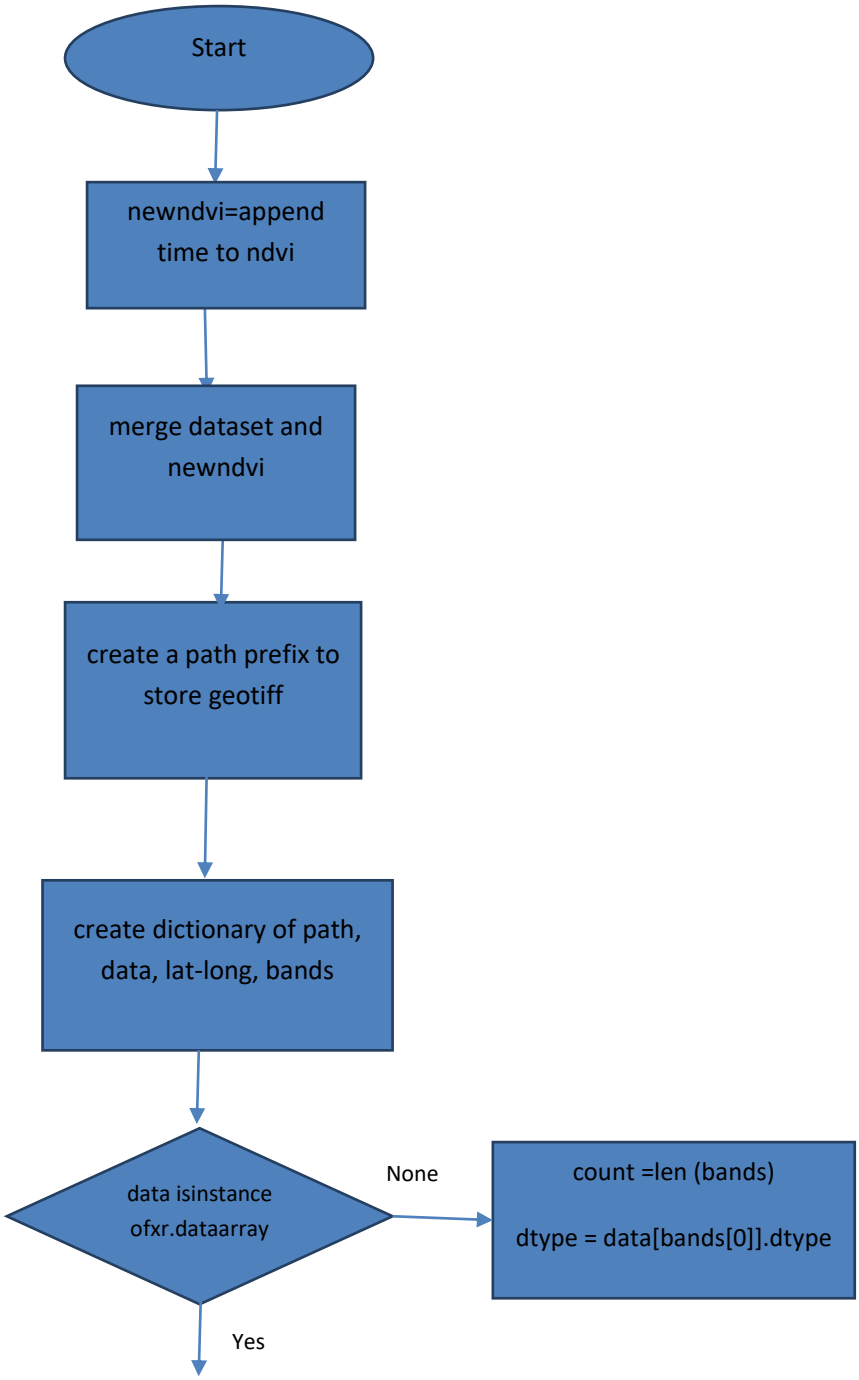


Fig 4.2: NDVI Calculation of a particular area. Values lie between 0-1. Values higher than 0 has healthy vegetation in those areas.

4.2 xArray to GeoTiff

All the calculations such as of spectral indices is performed on xArray datasets and the results are also in xArray datasets without time attribute, so there is a need to convert those results in Geotiff format for future use.



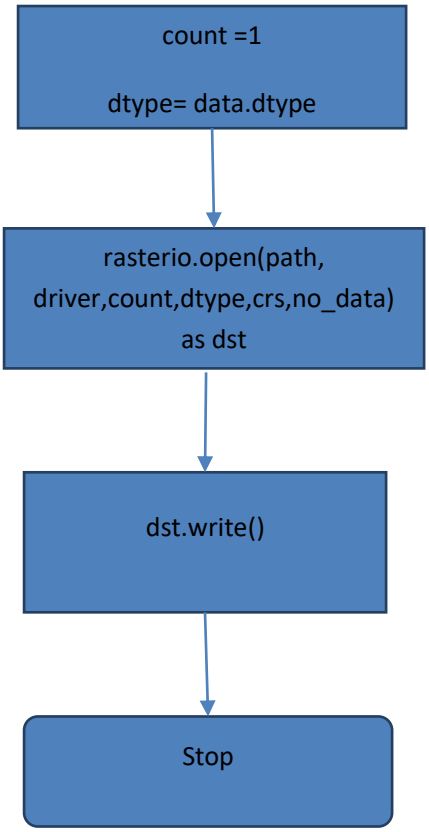


Fig 4.3: xArray to Geotiff Algorithm

NDVI xArray dataset contains only NDVI values for particular latitude-longitude bound. For the time series analysis there is a need to append time to that dataset. After it merging the sampled dataset and NDVI dataset will result in a xArray dataset containing information of all bands, crs value and ndvi values.

Next there is a need to create a path to store the geotiff file. After it, dictionary will be created which consist of latitude, longitude, bands, path and data attributes. Then we have to check whether data is instance of xArray dataset. If it returns none then count variable will be assigned the no. of all bands in the dataset and datatype named as dtype will be assigned as a datatype of the first band. If it returns yes then count will be assigned as1 and dtype as data dtype. With the help of rasterio we can write geotiff files after getting all the required information. Geotiff file must have crs value, resolution, data type of all bands and bands details.

Result:

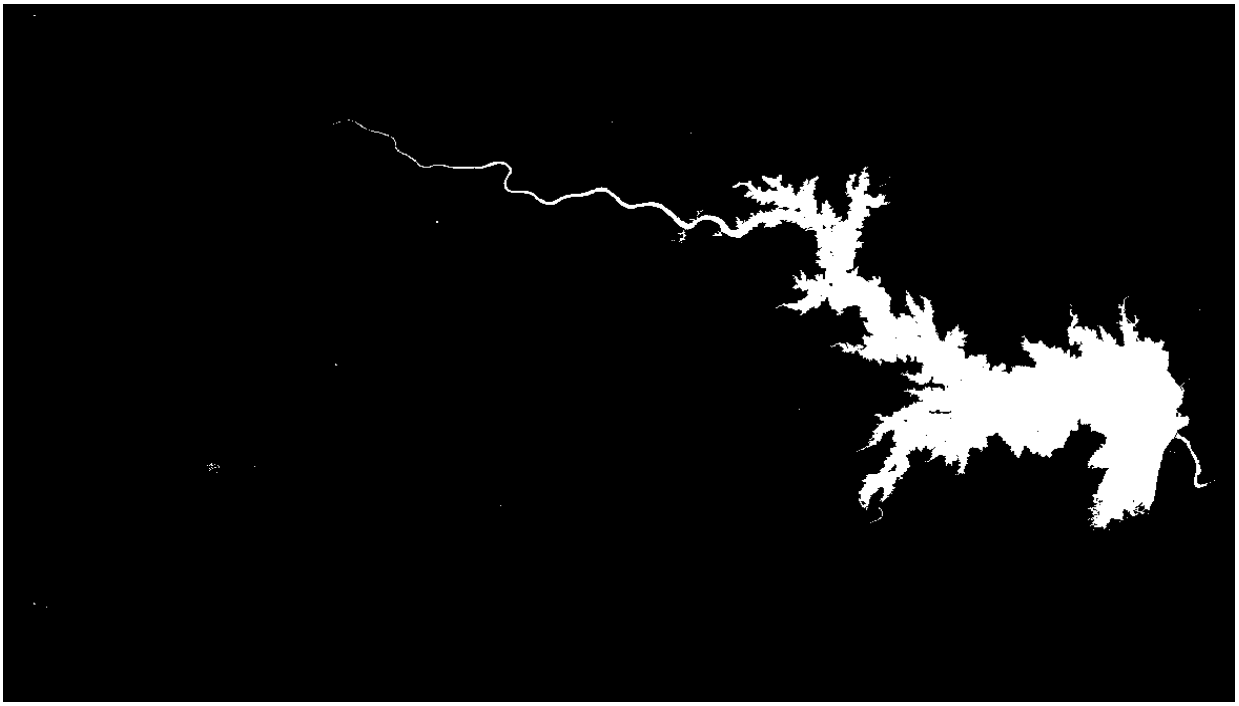


Fig 4.4: TIFF file of water detection at a particular area (xArray to Geotiff)

Fig 4.5: Shape (shp) file of water detection through QGIS Software

4.3 Time Series Plotting

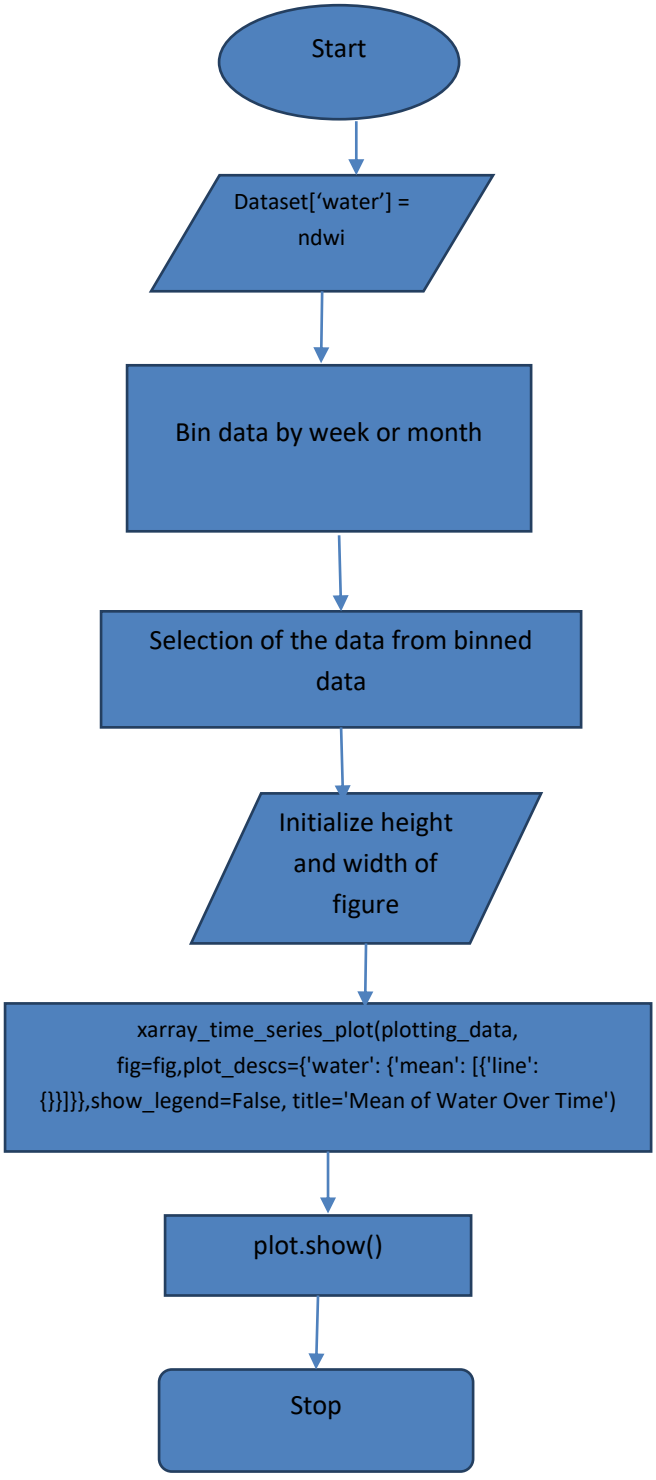


Fig 4.6: Time Series Analysis Algorithm

This algorithm will allow users to view the measured indices in time series. Here the spectral indices that we have used is NDWI and the algorithm will plot a graph based on the time series ingested data.

Results:

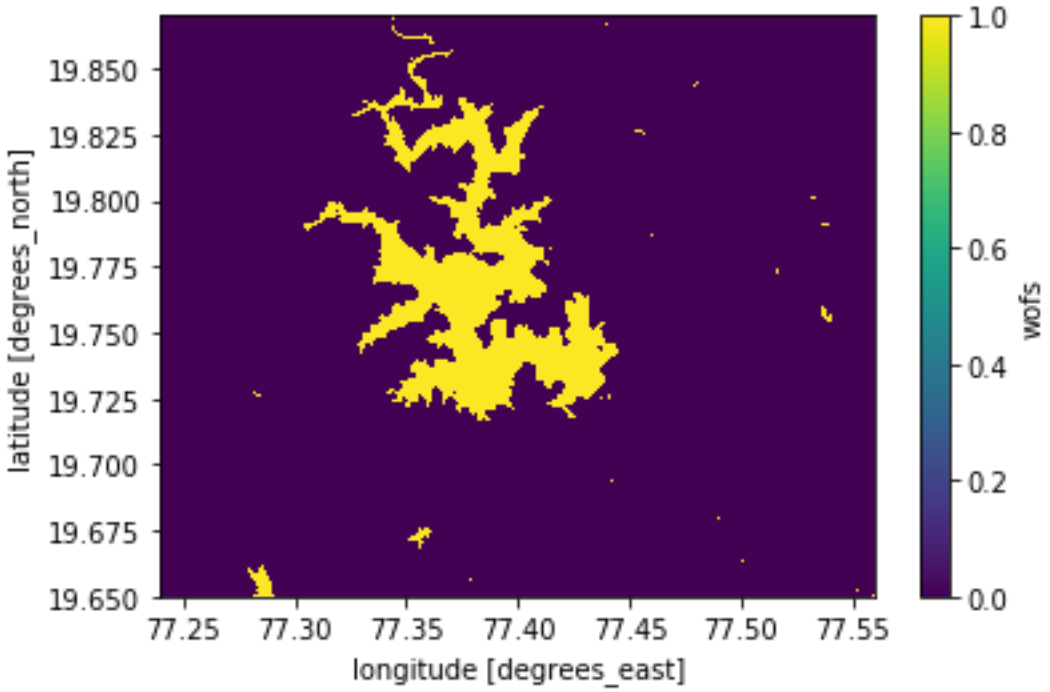


Fig 4.7: Water detection of a particular area (Landsat_5). Value1 indicates water

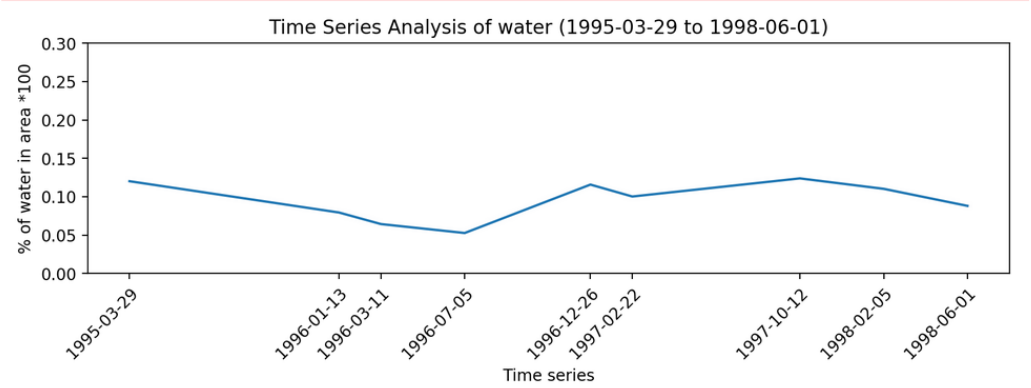


Fig 4.8: Time series analysis of water at a particular area

4.4 Crop detection using Random Forest Model

ESRI Shapefile has been used that contains training data of crops. Our first approach is to convert shapefile to geotiff through gdal library.

```
 # Import Python 3 print function

from __future__ import print_function

# Import OGR -

from osgeo import ogr

# Open the dataset from the file

dataset = ogr.Open('/Folder/.training_data.shp')

# Make sure the dataset exists -- it would be None if we couldn't open it

if not dataset:

print('Error: could not open dataset')
```

Just like GDAL, OGR abstracts the file formats.

```
#fetching properties of shapefile

### Let's get the driver from this file

driver = dataset.GetDriver()

print('Dataset driver is: {n}\n'.format(n=driver.name))

### How many layers are contained in this Shapefile?

layer_count = dataset.GetLayerCount()

print('The shapefile has {n} layer(s)\n'.format(n=layer_count))

### What is the name of the 1 layer?

layer = dataset.GetLayerByIndex(0)

print('The layer is named: {n}\n'.format(n=layer.GetName()))

### What is the layer's geometry? is it a point? a polyline? a polygon?

# First read in the geometry - but this is the enumerated type's value

geometry = layer.GetGeomType()

# So we need to translate it to the name of the enum
```

```
geometry_name = ogr.GeometryTypeToName(geometry)
print("The layer's geometry is: {geom}\n".format(geom=geometry_name))
### What is the layer's projection?
# Get the spatial reference
spatial_ref = layer.GetSpatialRef()
# Export this spatial reference to something we can read... like the Proj4
proj4 = spatial_ref.ExportToProj4()
print('Layer projection is: {proj4}\n'.format(proj4=proj4))
### How many features are in the layer?
feature_count = layer.GetFeatureCount()
print('Layer has {n} features\n'.format(n=feature_count))
### How many fields are in the shapefile, and what are their names?
# First we need to capture the layer definition
defn = layer.GetLayerDefn()
# How many fields
field_count = defn.GetFieldCount()
print('Layer has {n} fields'.format(n=field_count))
print('Their names are: ')
for i in range(field_count):
field_defn = defn.GetFieldDefn(i)
print('\t{name} - {datatype}'.format(name=field_defn.GetName(),
datatype=field_defn.GetTypeName()))
```

The training data we just opened contains two fields:

an ID field (Integer datatype)

a class field (String datatype)

Combined with the innate location information of polygons in a Shapefile, fields resemble all that we need to use for pairing labels (i.e., the integer ID and the string description) with the information in our raster.

However, in order to pair up our vector data with raster pixels, we will need a way of co-aligning the datasets in space.

One (complicated) way of doing this would be to manually loop through each polygon in our vector layer and determine which pixels from our raster are contained within. This approach is exactly what GIS softwares (e.g., ENVI, ArcGIS, QGIS) do when doing pairing rasters with vectors, like when doing zonal statistics.

Another less complicated way would be to use the concept of a Region of Interest (ROI) image where each non-zero pixel value in our ROI image corresponds to a raster representation of a polygon from our vector layer. In the example of our training data, most of the values would be 0 in the rasterized representation because our training data samples are small compared to the entire study area. Because we have assigned an integer ID field to each polygon, we could use these integers to store information about which polygons belong to which pixels.

```
# Print out metadata about raster

gdalinfo -proj4 '/Folder/LE70220491999322EDC01.tif
```

We will use Upperleft and Lower right coordinates which we get from the above command.

```
# Import GDAL

from osgeo import gdal

# First we will open our raster image, to understand how we will want to rasterize our vector

raster_ds = gdal.Open('/Folder/LE70220491999322EDC01_stack.tif',
gdal.GA_ReadOnly)

# Fetch number of rows and columns

ncol = raster_ds.RasterXSize

nrow = raster_ds.RasterYSize

# Fetch projection and extent

proj = raster_ds.GetProjectionRef()

ext = raster_ds.GetGeoTransform()
```

```
raster_ds = None

# Create the raster dataset

memory_driver = gdal.GetDriverByName('GTiff')

out_raster_ds = memory_driver.Create('/Folder/training_data.tif', ncol, nrow, 1,
gdal.GDT_Byte)

# Set the ROI image's projection and extent to our input raster's projection and extent

out_raster_ds.SetProjection(proj)

out_raster_ds.SetGeoTransform(ext)

# Fill our output band with the 0 blank, no class label, value

b = out_raster_ds.GetRasterBand(1)

b.Fill(0)

# Rasterize the shapefile layer to our new dataset

status = gdal.RasterizeLayer(out_raster_ds,  # output to our new dataset

                             [1],  # output to our new dataset's first band

                             layer,  # rasterize this layer

                             None, None,  # transformation as in same projection

                             [0],  # burn value 0

                            [ 'ATTRIBUTE=id'] ) # put raster values according to the
'id

 # Close dataset

out_raster_ds = None
```

To classify the Landsat image supervised classification approach is used which incorporates the training data. Specifically, RandomForest ensemble decision tree algorithm has been used. The RandomForest algorithm has recently become extremely popular in the field of remote sensing, and is quite fast when compared to some other machine learning approaches (e.g., SVM can be quite computationally intensive).

A brief explanation of the RandomForest algorithm comes from the name. Rather than utilize the predictions of a single decision tree, the algorithm will take the ensemble result of a large number of decision trees (a forest of them). The "Random"

part of the name comes from the term "bootstrap aggregating", or "bagging". What this means is that each tree within the forest only gets to train on some subset of the full training dataset (the subset is determined by sampling with replacement). The elements of the training data for each tree that are left unseen are held "out-of-bag" for estimation of accuracy. Randomness also helps decide which feature input variables are seen at each node in each decision tree. Once all individual trees are fit to the random subset of the training data, using a random set of feature variable at each node, the ensemble of them all is used to give the final prediction.

In the classification mode, this means that if we have 8 classes being predicted using 500 trees, the output prediction would be the class that has the most number of the 500 trees predicting it. The proportion of the number of trees that voted for the winning class can be a diagnostic of the representativeness of your training data relative to the rest of the image. Taking the 500 trees example, if we have pixels which are voted to be in the "Wheat" land cover class by 475 of 500 trees, wewould say that this was a relatively certain prediction. On the other hand, if we have a pixel which gets 250 votes for "Wheat" and 225 votes for "Tobacco", we could interpret this as either an innately confusing pixel (maybe it is a mixed pixel, or it is a small area of wheat) or as an indicator that we need more training data samples in these types of pixels.

We have used Random Forest Model provided by Scikit-learn library. Scikit-learn is a machine learning library that provides easy and consistent interfaces to many of the most popular machine learning algorithms. It is built on top of the pre-existing scientific Python libraries, including NumPy, SciPy, and matplotlib, which makes it very easy to incorporate into your workflow.

In the first step image reading and ROI image reading has been done.

```
from __future__ import print_function, division

# Import GDAL, NumPy, and matplotlib

from osgeo import gdal, gdal_array

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

# Tell GDAL to throw Python exceptions, and register all drivers

gdal.UseExceptions()

gdal.AllRegister()
```

```
# Read in our image and ROI image

img_ds = gdal.Open('/Folder/LE70220491999322EDC01_stack.tif',
gdal.GA_ReadOnly)

roi_ds = gdal.Open('/Folder/training_data.tif', gdal.GA_ReadOnly)

img = np.zeros((img_ds.RasterYSize, img_ds.RasterXSize, img_ds.RasterCount),
gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBand(1).DataTy
pe))for b in range(img.shape[2]):

img[:, :, b] = img_ds.GetRasterBand(b + 1).ReadAsArray()

roi = roi_ds.GetRasterBand(1).ReadAsArray().astype(np.uint8)

# Display them

plt.subplot(121)

plt.imshow(img[:, :, 4], cmap=plt.cm.Greys_r)

plt.title('SWIR1')

plt.subplot(122)

plt.imshow(roi, cmap=plt.cm.Spectral)

plt.title('ROI Training Data')

plt.show()

roi_ds = gdal.Open('output/geotiffs/ndwi_1996_02_24_04_27_40.tif',
gdal.GA_ReadOnly)
```

We have the image we want to classify (our X feature inputs), and the ROI with the land cover labels (our Y labeled data), we need to pair them up in NumPy arrays so we may feed them to Random Forest:

```
# Find how many non-zero entries we have -- i.e. how many training data samples?

n_samples = (roi> 0).sum()

print('We have {n} samples'.format(n=n_samples))

# What are our classification labels?

labels = np.unique(roi[roi> 0])

print('The training data include {n} classes: {classes}'.format(n=labels.size,
```

```
                                    classes=labels))
```

We will need a "X" matrix containing our features, and a "y" array containing our labels. These will have n_samples rows. In other languages we would need to allocate these and them loop to fill them, but NumPy can be faster.

```
X = img[roi> 0, :]

y = roi[roi> 0]

print('Our X matrix is sized: {sz}'.format(sz=X.shape))

print('Our y array is sized: {sz}'.format(sz=y.shape))
```

Mask out clouds, cloud shadows, and snow using Fmask

```
clear = X[:, 7] <= 1

X = X[clear, :7]

y = y[clear]

print('After masking, our X matrix is sized: {sz}'.format(sz=X.shape))

print('After masking, our y array is sized: {sz}'.format(sz=y.shape))
```

We have X matrix of feature inputs (the spectral bands) and y array (the labels), we can train the model.

```
from sklearn.ensemble import RandomForestClassifier

# Initialize our model with 500 trees

rf = RandomForestClassifier(n_estimators=500, oob_score=True)

# Fit our model to training data

rf = rf.fit(X, y)
```

With Random Forest model fit, we can check out the "Out-of-Bag" (OOB) prediction score:

```
print('Our OOB prediction of accuracy is: {oob}%'.format(oob=rf.oob_score_ * 100))
```

Output of OOB prediction score is **95.67 %** which is good enough.

#import pandas library for cross-tabulation to see confusion-matrix.

```
import pandas as pd
```

```
# Setup a dataframe -- just like R
df = pd.DataFrame()
df['truth'] = y
df['predict'] = rf.predict(X)
# Cross-tabulate predictions
print(pd.crosstab(df['truth'], df['predict'], margins=True))
```

With Random Forest classifier fit, we can now proceed by trying to classify the entire image:

```
# Take our full image, and reshape into long 2d array (nrow * ncol, nband) for classification
new_shape = (img.shape[0] * img.shape[1], img.shape[2] - 1)
img_as_array = img[:, :, :7].reshape(new_shape)
print('Reshaped from {o} to {n}'.format(o=img.shape,
                                        n=img_as_array.shape))

# Now predict for each pixel
class_prediction = rf.predict(img_as_array)
# Reshape our classification map
class_prediction = class_prediction.reshape(img[:, :, 0].shape)
#Now will convert the array to raster to view the results
import gdal, ogr, os, osr
import numpy as np
def array2raster(newRasterfn,rasterOrigin,pixelWidth,pixelHeight,array):
    cols = array.shape[1]
    rows = array.shape[0]
originX = rasterOrigin[0]
originY = rasterOrigin[1]
```

```
driver = gdal.GetDriverByName('GTiff')
outRaster = driver.Create(newRasterfn, cols, rows, 1, gdal.GDT_Byte)
outRaster.SetGeoTransform((originX, pixelWidth, 0, originY, 0, pixelHeight))
outband = outRaster.GetRasterBand(1)
print(outband)
outband.WriteArray(array)
outRasterSRS = osr.SpatialReference()
outRasterSRS.ImportFromEPSG(4326)
outRaster.SetProjection(outRasterSRS.ExportToWkt())
outband = None
#Calling array2raster
rasterOrigin = (0,0)
pixelWidth = 1
pixelHeight = -1
newRasterfn = 'Folder/arraytif.tif'
array = class_prediction
array2raster(newRasterfn,rasterOrigin,pixelWidth,pixelHeight,array)
```

Results:

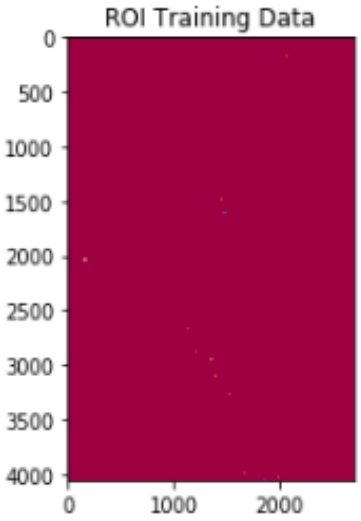


Fig 4.9: Training data (Geotiff format)

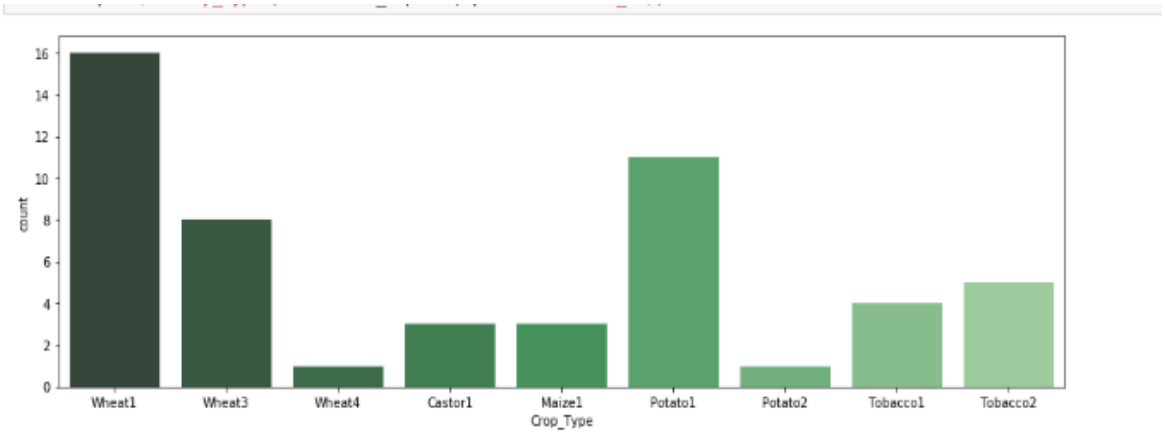


Fig 4.10: Crop Type in training data

Fig4.11: Test data example

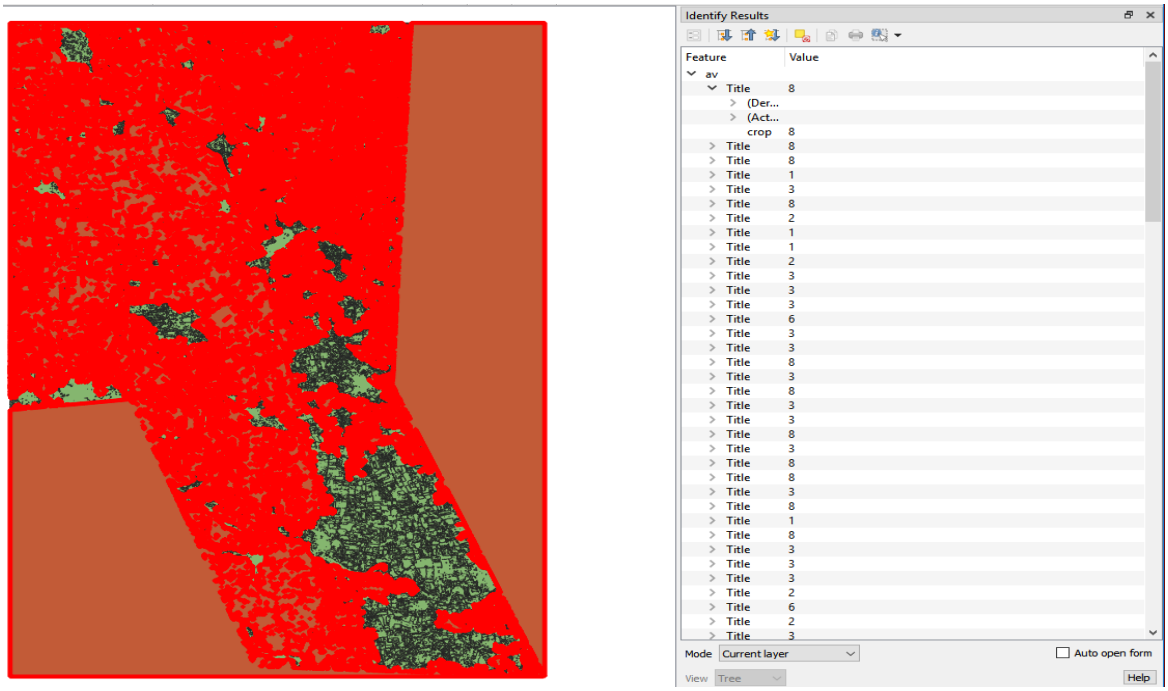


Fig 4.12: Result dataset shows area of detected crops

predict	1	2	3	6	8	All
truth						
1	2629	0	0	0	0	2629
2	2	896	0	0	0	898
3	0	0	604	0	0	604
6	1	0	0	396	0	397
8	0	0	0	0	1723	1723
All	2632	896	604	396	1723	6251

Fig 4.13: Cross-Tabulation Matrix: Predicts 5 crops out of 8 crops in the specific area

CHAPTER-5

SCREENSHOTS

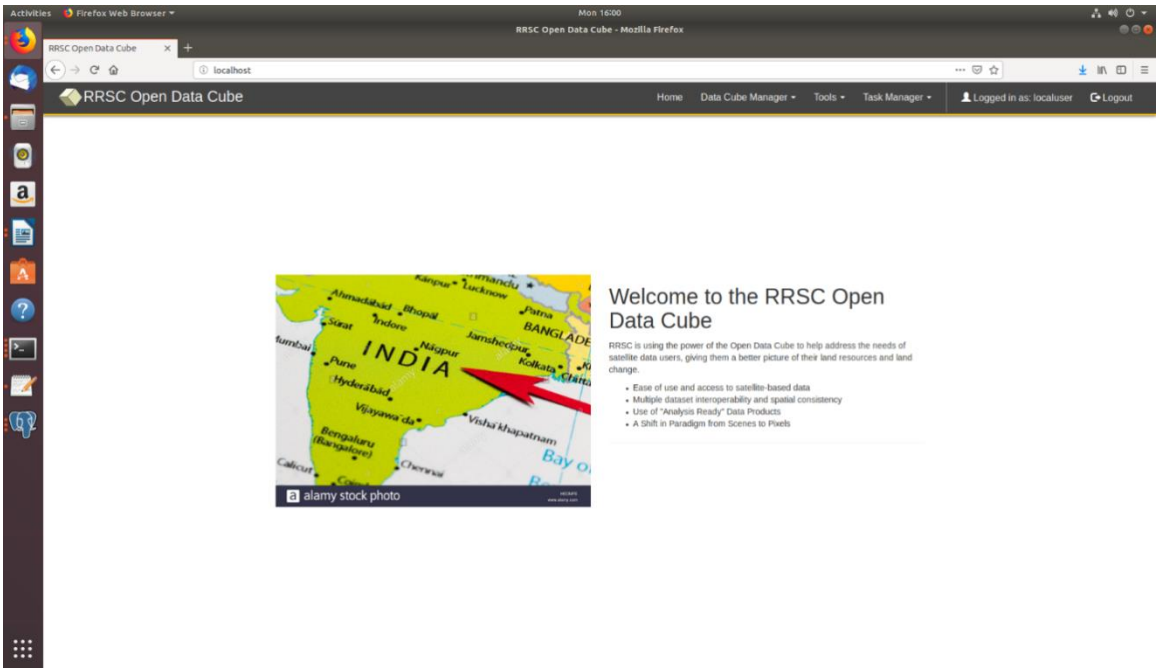


Fig 5.0.1: ODC UI Home

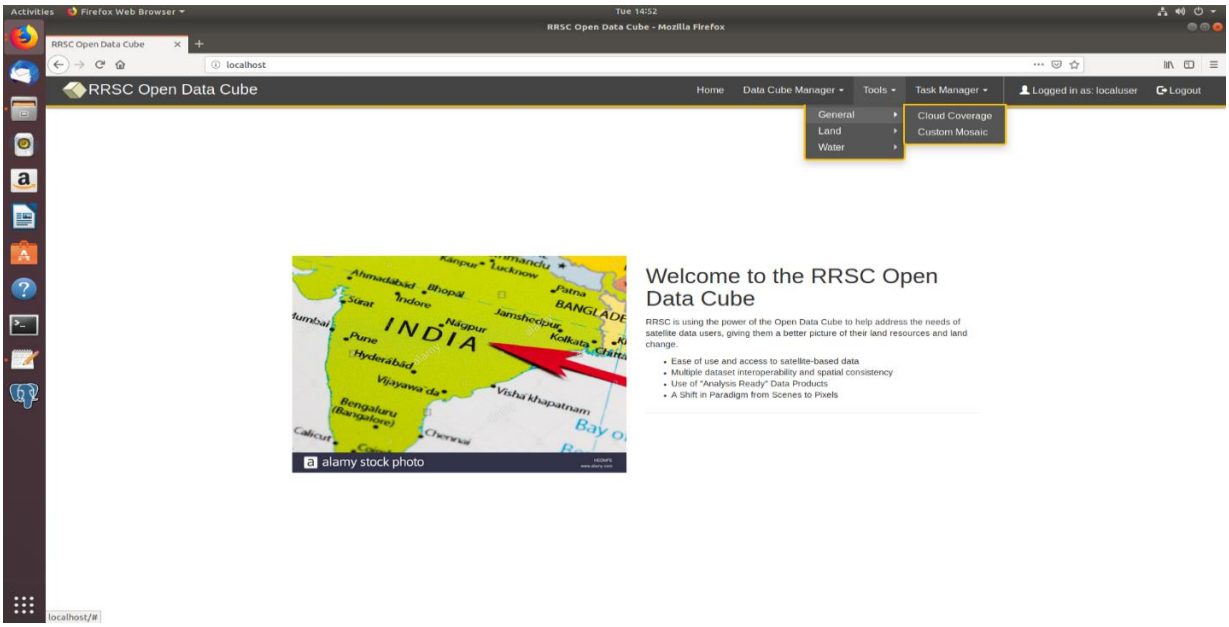


Fig 5.2: ODC UI Tool

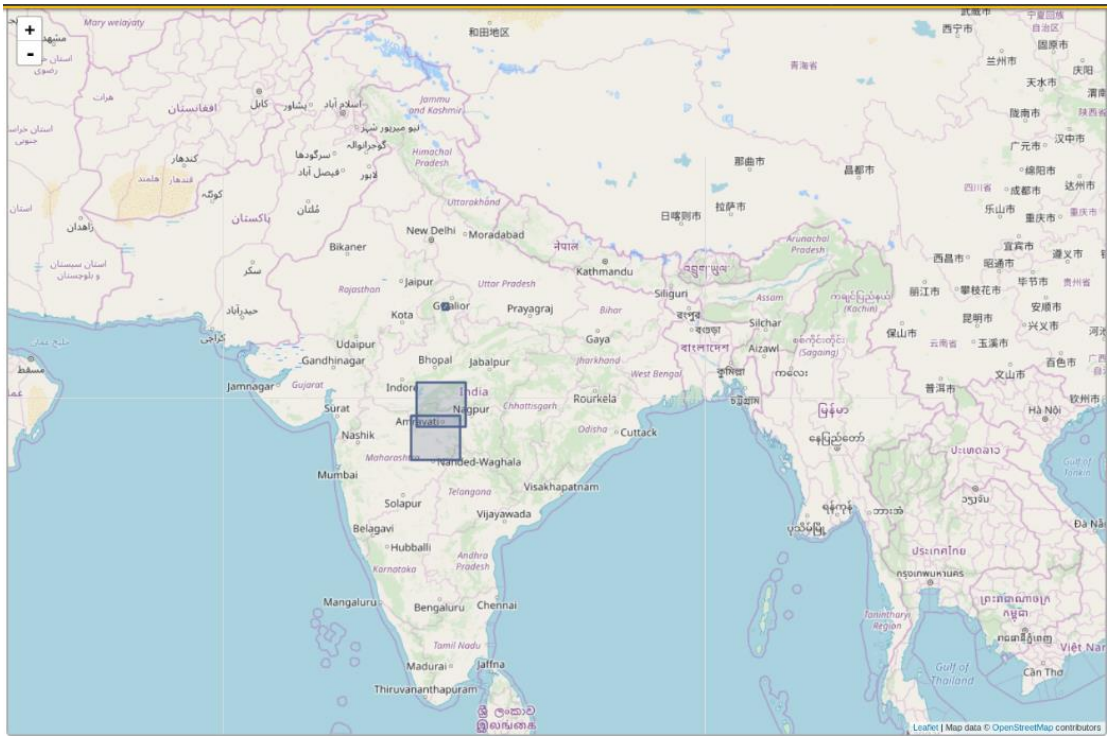


Fig 5.3: Data Cube Visualization (3 cubes)

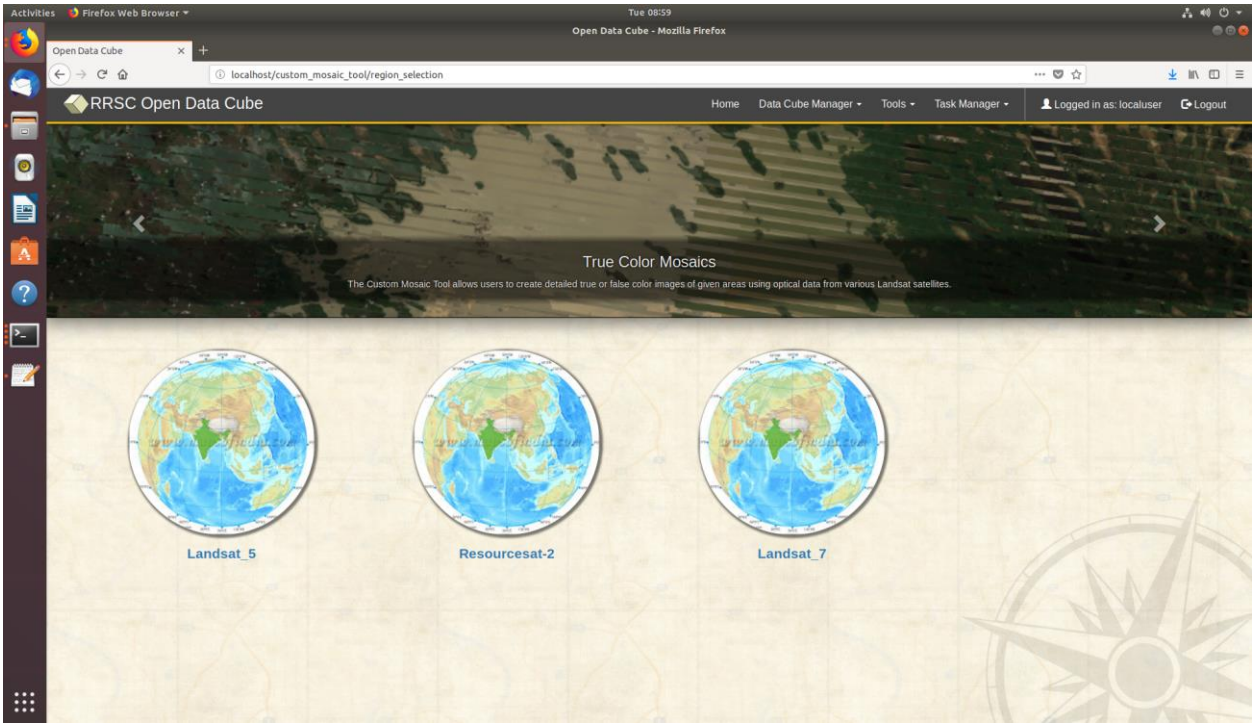


Fig 5.4: Ingested Datasets in ODC UI

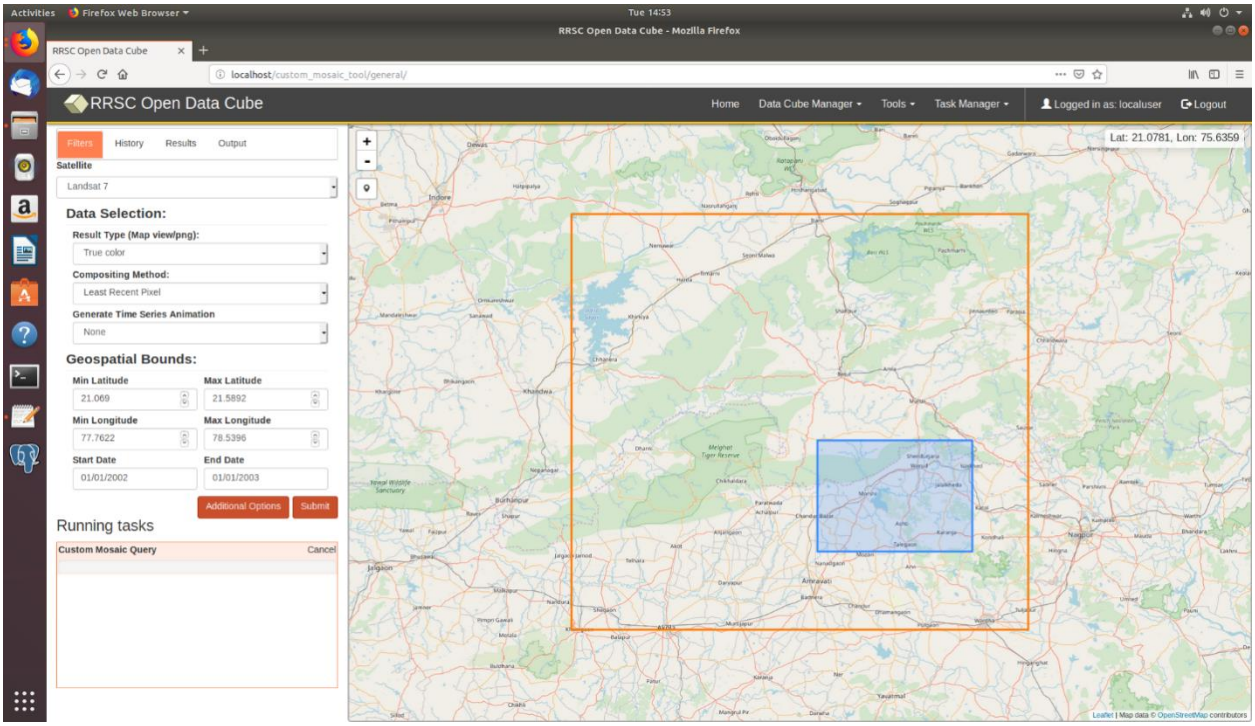


Fig 5.5: Selection of area and time under the specific tool for the selected ingested dataset

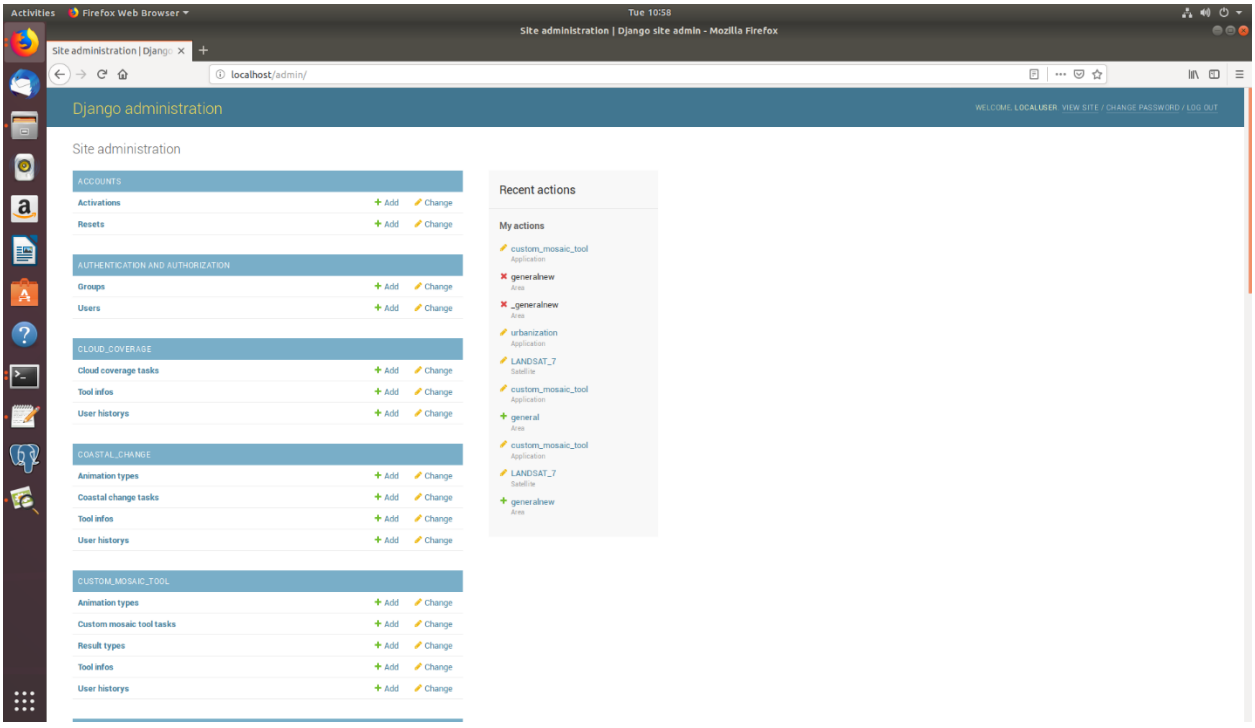


Fig 5.6: Open Data Cube UI Admin

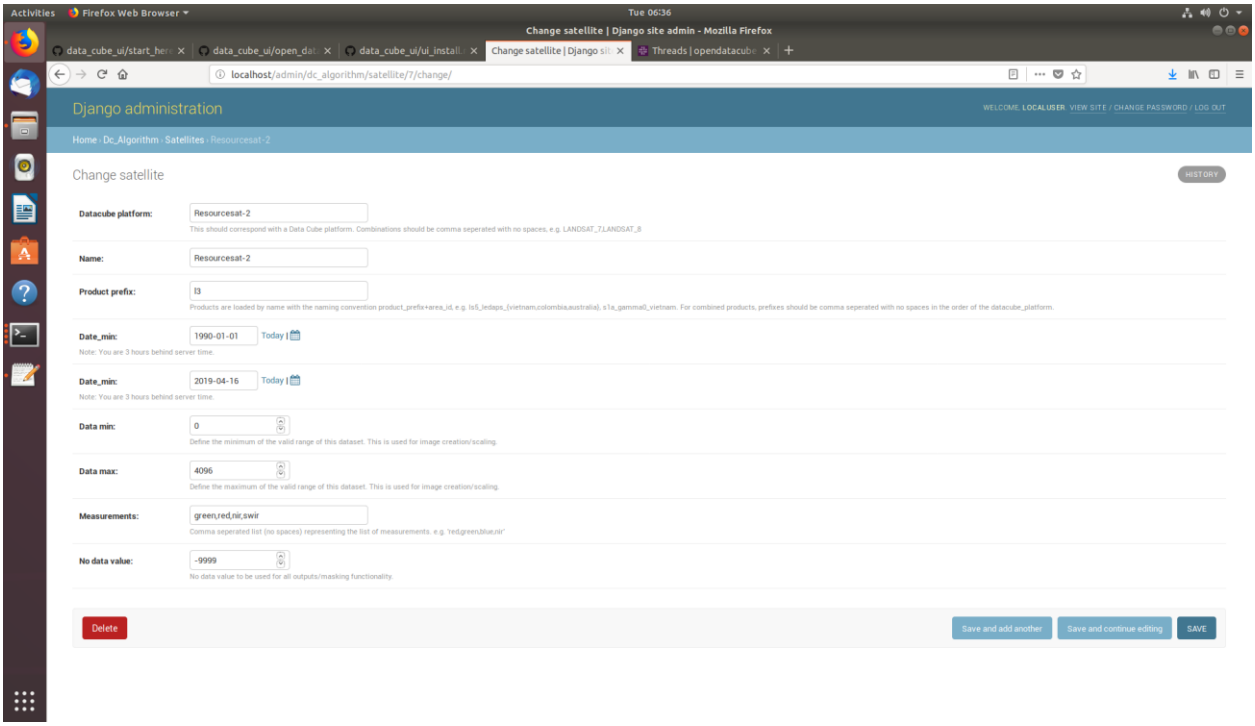


Fig 5.7: Open Data Cube UI Admin Satellite addition

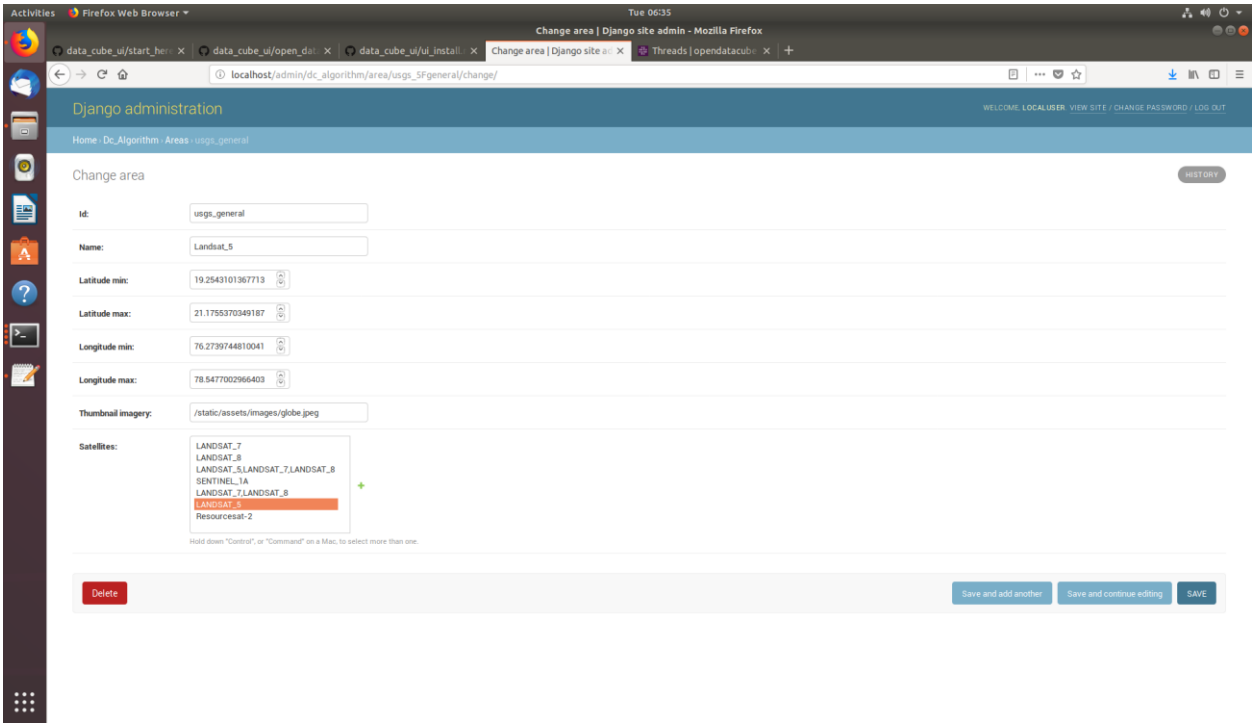


Fig 5.8: Open Data Cube UI Admin Area Addition

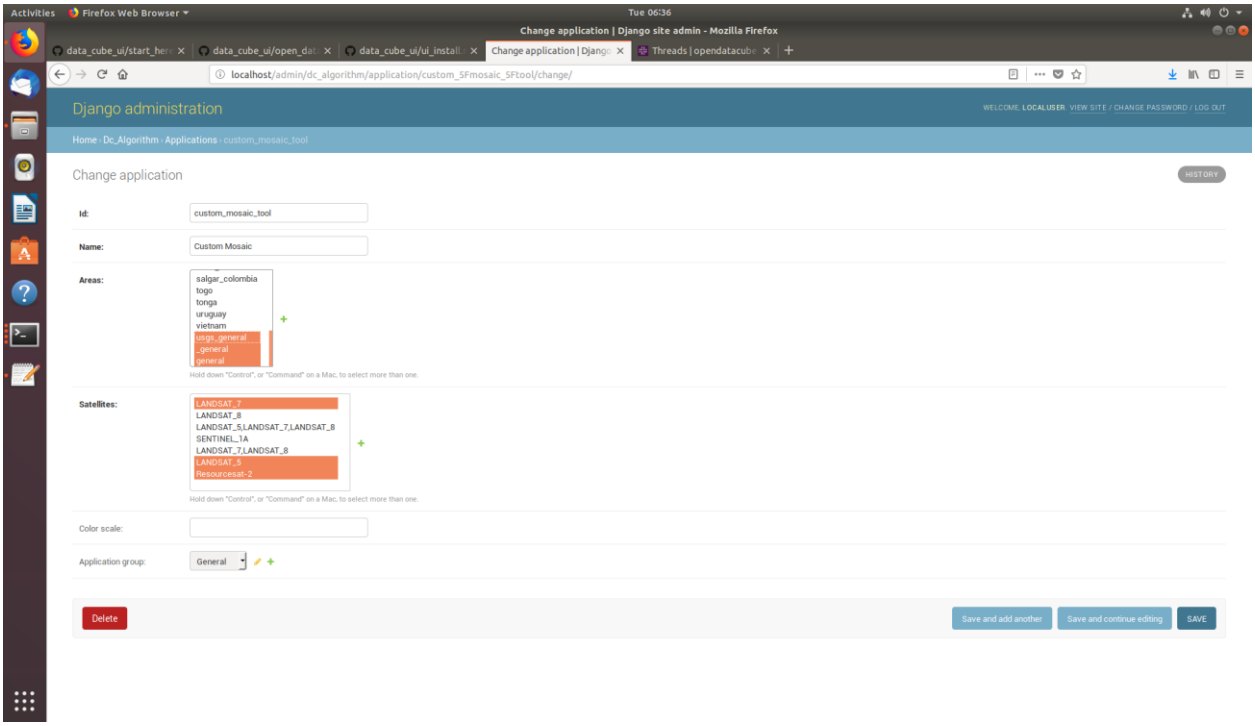


Fig 5.9: Addition of area in particular application

CHAPTER-6

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

As new generations of EO data create increasingly larger volumes of data, global users will struggle to prepare and manage that data to address their local, regional and national decision-making needs. The ODC initiative provides an innovative, free and open data architecture solution that lowers the technical barrier for global users to exploit satellite data and optimize its societal benefit.

Array database technique provides flexible and scalable storage of multidimensional array data, which is necessary to store hundreds of terabytes of satellite data. With the Data cube technology these arrays can be arranged in the form of cubes. Developed web application designed for scientists which contains all the details about the ingested datasets and tools which provides better analysis of stored data whose results can be further used for scientific purpose.

5.2 Future Scope

For the better observation of Earth data, so that relevant steps can be taken after it the developed application will be implemented on a cluster computing environment which will provide good performance in time series data analysis. New algorithms will be developed and can be added in the UI so that the user can take benefit of it

Future work will incorporate the advancement of Data 3D shape for Indian information, creating calculations that can bolster Indian satellite information. Time series analysis of Indian information will turn out to be more successful for investigation and close outcomes on it.

References

i. Zhenyu Tan, Peng Yue, and Jianya Gong, *An Array Database Approach for Earth Observation Data Management and Processing*. ISPRS International Journal of Geo-Information, 2017.

ii. Martin Sudmanns, Stefan Lang, Dirk Tiede, *Big Earth Data: From Data to Information*. GISCIENCE 2018.

iii. Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*. Google, Inc. 2005,

iv. Brian Killough, *Overview of the Open Data Cube Initiative*. (IGARSS) IEEE International Geoscience and Remote Sensing Symposium, 2018.

v. Argyro Kavvada and Alex Held, *Analysis-Ready Earth Observation Data And The United Nations Sustainable Development Goals* . IGARSS 2018

vi. Peter Strobl, Peter Baumann, Adam Lewis, Zoltan Szantoi, Brian Killough, Matthew Purss, Max Craglia, Stefano Nativi1, Alex Held and Trevor Dhu, *The Six Faces Of The Data Cube*. Big Data from Space (BiDS'17).

vii. Michael F. Goodchild1 and Robert P. Haining, *Gis And Spatial Data Analysis: Converging Perspectives*. National Center for Geographic Information and Analysis, 2017.

viii. P.L.N. Raju, *Spatial Data Analysis*. Satellite Remote Sensing and GIS Applications in Agricultural Meteorology, 2017.

ix. Alex Leith, "Open Data Cube", 2018. . [Online]. Available:https://medium.com/opendatacube/what-is-open-data-cube-805af60820d7

x. AdamLewisa, SimonOlivera, LeoLymburnera, BenEvansb, LesleyWyborn, NormanMueller, GregoryRaevksi, JeremyHooke, RobWoodcock, JoshuaSixsmith, WenjunWu, PeterTan, FuqinLi, BrianKillough, StuartMinchin, DaleRoberts, DamienAyers, BiswajitBalaa, Lan-WeiWanga, *The Australian Geoscience Data Cube — Foundations and lessons learned*. ELSEVIER- Remote Sensing of Environment, 2017.

xi. Peter Baumann, DimitarMisev, Vlad Merticariu, Bang Pham Huu, Brennan Bell, *Datacubes: A Technology Survey*, IGARSS 2018.

xii. Open Data Cube *White Paper.*
Available:http://ceos.org/document_management/Meetings/SIT/SIT-32/Side%20Meeting%20Materials/ODC_WhitePaper_v2a.pdf

xiii. *"*Open Data Cube*", opendatacube.org.* [Online].
Available :https://www.opendatacube.org/

xiv. "GDAL", *gdal.org.* [Online].
Available :https://www.gdal.org/

xv. "Australian Geo Science Data Cube", *Analysis Ready Data, Ingestion and Indexing.* [Online].
Available :https://en.wikipedia.org/wiki/Australian_Geoscience_Data_Cub

xvi. "Geographic Information System", [Online].
Available:https://grindgis.com/blog/pros-and-cons-of-gis-geographic-information-system

xvii. Shivaprakash Yaragal, *Big data in GIS environment.* [Online].
Available:https://www.geospatialworld.net/blogs/big-data-in-gis-environment/

xviii. "POSTGRESQL", [Online].
Available :https://www.postgresql.org/