Dr. K. S. Rajan

Dr. Kiran Chand

Dr. Kuldeep Kurte

Malini Krishnan

Mewan War

Sourabh Barala

Workshop on

Zarr for ARD and Geodatacubes – A Use case for Analyzing Forest Fires

by

Lab for Spatial Informatics, IIIT-Hyderabad

LAB FOR SPATIAL INFORMATICS (LSI)

LSI Research Center

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

About

❖LSI@IIIT Hyderabad has been at the forefront of Geospatial Science, Research & Applications with a multi-disciplinary approach.

❖All facets of Geospatial Sciences - study, analysis and modelling of space-time characteristics of an object, event or phenomena; combined with evolving technological landscape - Sensor tech like satellite, drone, LiDAR, IoT for data acquisition; Geodata storage & handling; Online Map Production/Dissemination, Visualization in 2D, 3D & 4D; and Application specific Solution & Services; are posing a range of research challenges.

❖The Research team @LSI works on these across Remote Sensing, Geospatial Data Sciences, and Modelling & Simulation of Land, Water and Environmental changes.

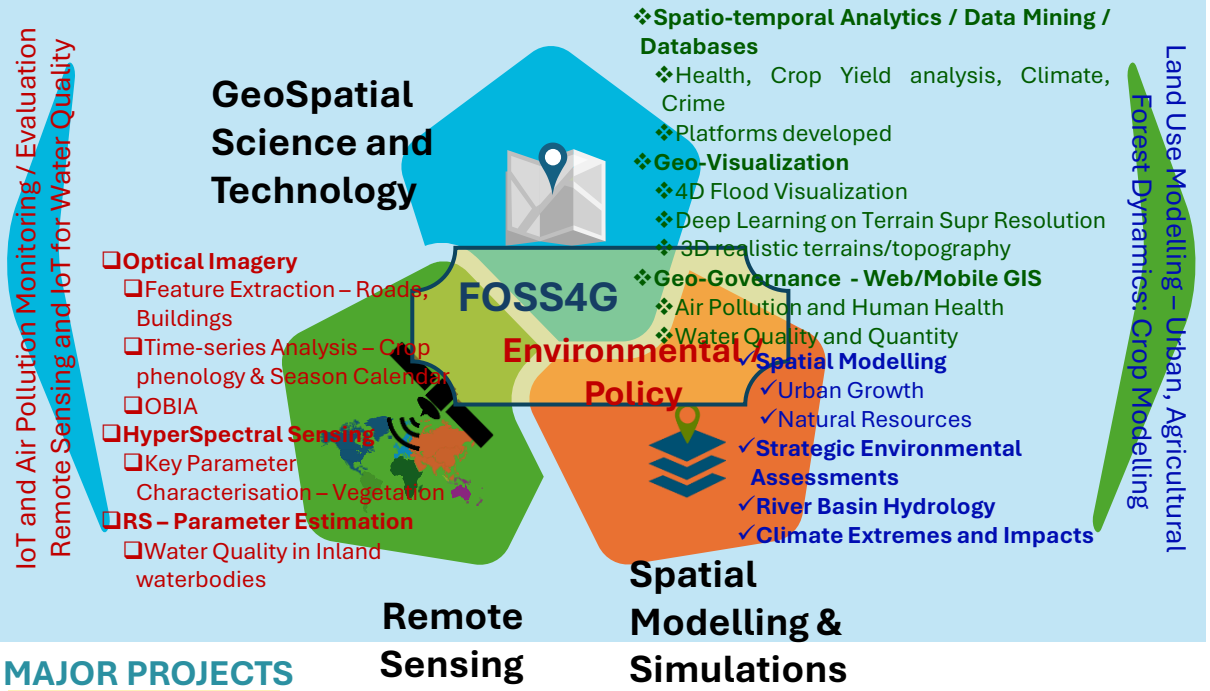
❖LSI provides a unique platform for fellow researchers to collaborate and to Post-Doc/PhD/MS level to work across core and applied research problems with a multi-disciplinary approach.

NEW - UG Dual Degree - CGD

❖A 5-year undergraduate Dual Degree program – BTech (Hons.) in computer Science & M.S. in Geospatial Technology (CGD) was launched from 2024-'25

❖Inspired by National Geospatial Policy 2022 that emphasizes high quality professionals and building geospatial domain research.

Current Research Landscape



MAJOR PROJECTS

❖Agri Monitored Reengineering and Transformation. DMF Sundargarh, Govt. of Odisha. 2021-26

❖Digital Door Numbering (DDN) Project. NIUM, Govt. of Telangana. 2021 onwards.

❖Modelling Extreme Risk of Water Quality under Hydro-Climatic Variability for Major River Basins of India. CRG SERB, DST (GoI). 2021-2024

❖Extraction of Objects of interest from LiDAR data - 3D Mesh Segmentation and Attribution. M/s CYIENT Ltd. 2021-23

❖Decision Support System and Software Development for Realtime Water Quantity and Quality Management of Tunga-Bhadra River System of Tropical India. TMD WTI, DST (GoI). 2021-24

❖IoT Enabled Smart Cities Pollution, Health and Governance. NRDMS, DST (GoI). 2020-23

Faculty at LSI receive research and project funding from both National and State Govt., Research Bodies and Industry.

PUBLICATIONS DURING LAST 3 YEARS - *https://lsi.iiit.ac.in/publications/*

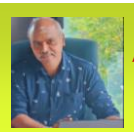
❖Journals – 38; Conferences - 14

Collaborations

IIT Tirupati Navavishkar I-Hub Foundation
OSGeo-IN
भारतीय विमानपत्तन प्राधिकरण AIRPORTS AUTHORITY OF INDIA Civil Aviation Research Organisation
इसरो isro nrsc
ICRISAT
ADVANCED CENTRE FOR INTEGRATED WATER RESOURCES MANAGEMENT

FACULTY

Prof. K. S. Rajan
Professor
rajan@iiit.ac.in
Geospatial Technologies & its applications.
LSI, EERC, COS



Dr. P Rama Chandra Prasad
Associate Professor
rcprasad@iiit.ac.in
Geospatial tools, Forest Ecosystem, Wetlands, LULC.
LSI, EERC

Dr. Shaik Rehana
Associate Professor
rehana.s@iiit.ac.in
Hydrology, Climate Change, Water Quality Management.
LSI, EERC

Dr. T.R. Kiran Chand
Associate Professor
Kiranchand.tr@iiit.ac.in
Applied Remote Sensing, Fire & Climate Change, Earth System Science, Sustainable

Dr. Kuldeep Ramachandra Kurte
Assistant Professor
Kuldeep.kurte@iiit.ac.in
Geosymantecs
Remote Sensing
LSI

https://www.iiit.ac.in https://lsi.iiit.ac.in

OSGeo

FOSS4G ASIA 2026 NASHIK

Session	Duration
Theory (ARD, Datacubes, Zarr)	9:00 – 9:30
Prepared Demo	9:30 – 9:45
Software Setup & Fixes	9:45 – 10:30
Hands-on (Forest Fire Dataset)	10:30 – 11:00
Tea Break	11:00 – 11:30
Hands –on (Continued)	11:30 – 12:30
Experiment with different datasets	12:30 – 01:00
Q&A, Conclusion	01:00-01:30
Total	240 min

Workshop Resources
hosted at:

Lab for Spatial Informatics, IIIT Hyderabad

https://lsi.iiit.ac.in/workshops/

Lab for Spatial Informatics

Home Project/Research Publications People Programme

WORKSHOPS

FOSS4G Asia 2026: Zarr for ARD and Geodatacubes – A Use case for Analyzing Forest Fires

This workshop presents a practical introduction to building and using Analysis-Ready Data (ARD) and geodatacubes to address the growing challenges posed by the rapid expansion of Earth Observation (EO) data. As EO datasets increase in volume and heterogeneity across sensors, resolutions, and time scales, users often spend excessive effort on preprocessing before meaningful analysis can begin. The workshop highlights how ARD and N-dimensional geodatacubes overcome these barriers by harmonizing diverse datasets into a uniform, analysis-ready structure. Particular emphasis is placed on the Zarr storage format, which is well suited for geospatial datacubes due to its scalability, cloud compatibility, compression, and efficient parallel I/O, enabling faster access and reduced computational burden for end users. Over a 240-minute, hands-on format, participants will learn core concepts of ARD and geodatacubes, explore Zarr-based datacube creation and access using Python, and practice standardized data dissemination using OGC-API implementations such as pygeoapi, including OLAP operations like slicing, dicing, and roll-up. A forest fire use case runs throughout the workshop to demonstrate real-world relevance, showing how multi-mission satellite data and weather or climate records can be integrated and analyzed to study fire occurrence, intensity, burnt areas, and associated environmental drivers. By grounding technical concepts in this societally important application, the workshop illustrates how ARD geodatacubes can support fire danger assessment, mitigation, and management. By the end, participants will be equipped to convert spatiotemporal datasets from their own domains into Zarr-based geodatacubes and disseminate them through standard, interoperable interfaces.

Speakers: Dr. K. S. Rajan, Dr. Kuldeep Kurte, Dr. Kiran Chand

Student Presenters: Mewan War, Malini Krishnan

Date: 22-Jan-2026

Venue: MIDC Cluster, Nashik, Maharashtra

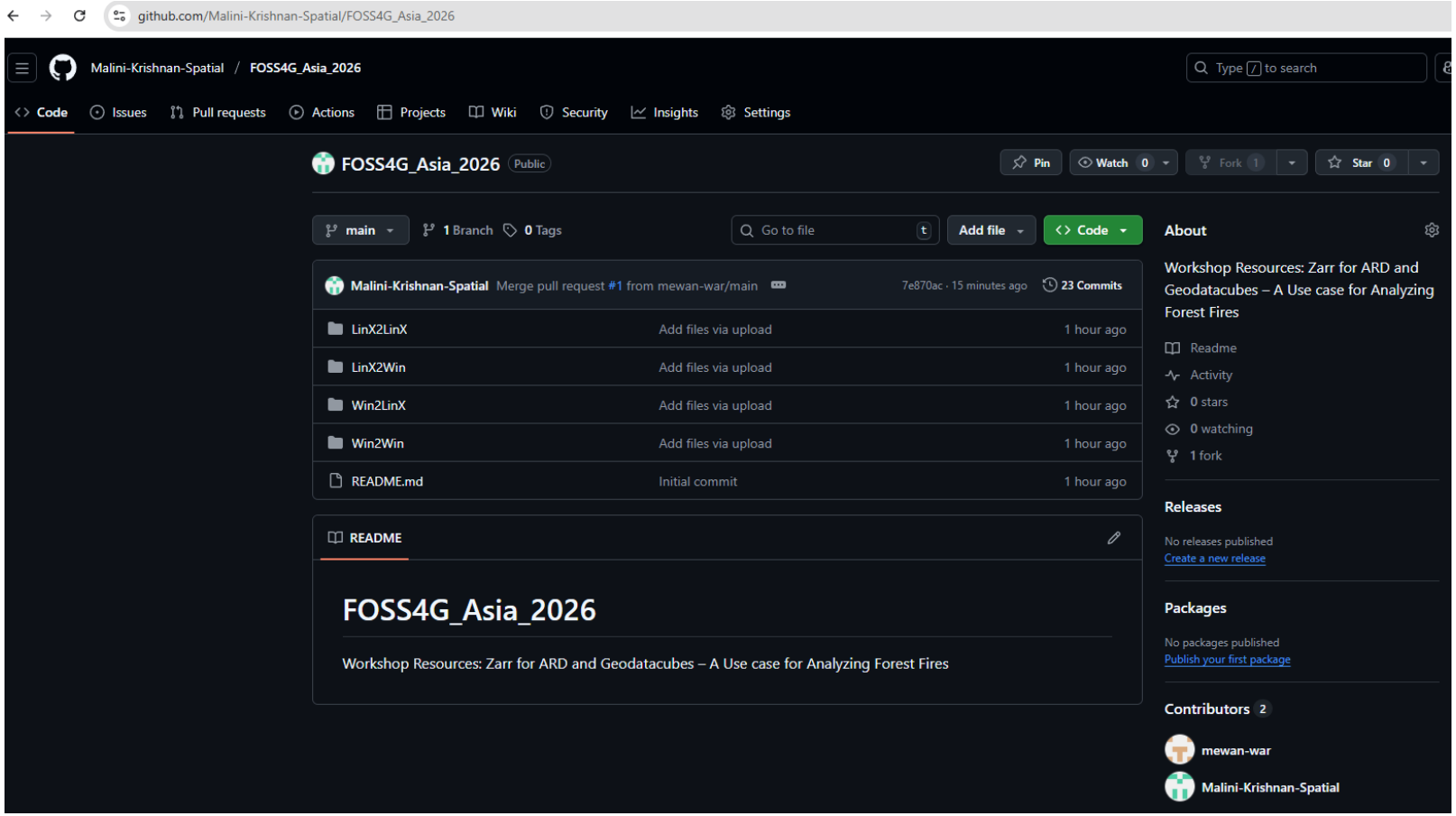
Workshop Resources: https://github.com/Malini-Krishnan-Spatial/FOSS4G_Asia_2026

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI Research Center

GitHub page with scripts and support files

https://github.com/Malini-Krishnan-Spatial/FOSS4G_Asia_2026



Operating System - Requisites

- Windows 7/10/11 - Any Edition
- Ubuntu Linux – Recent versions, preferred

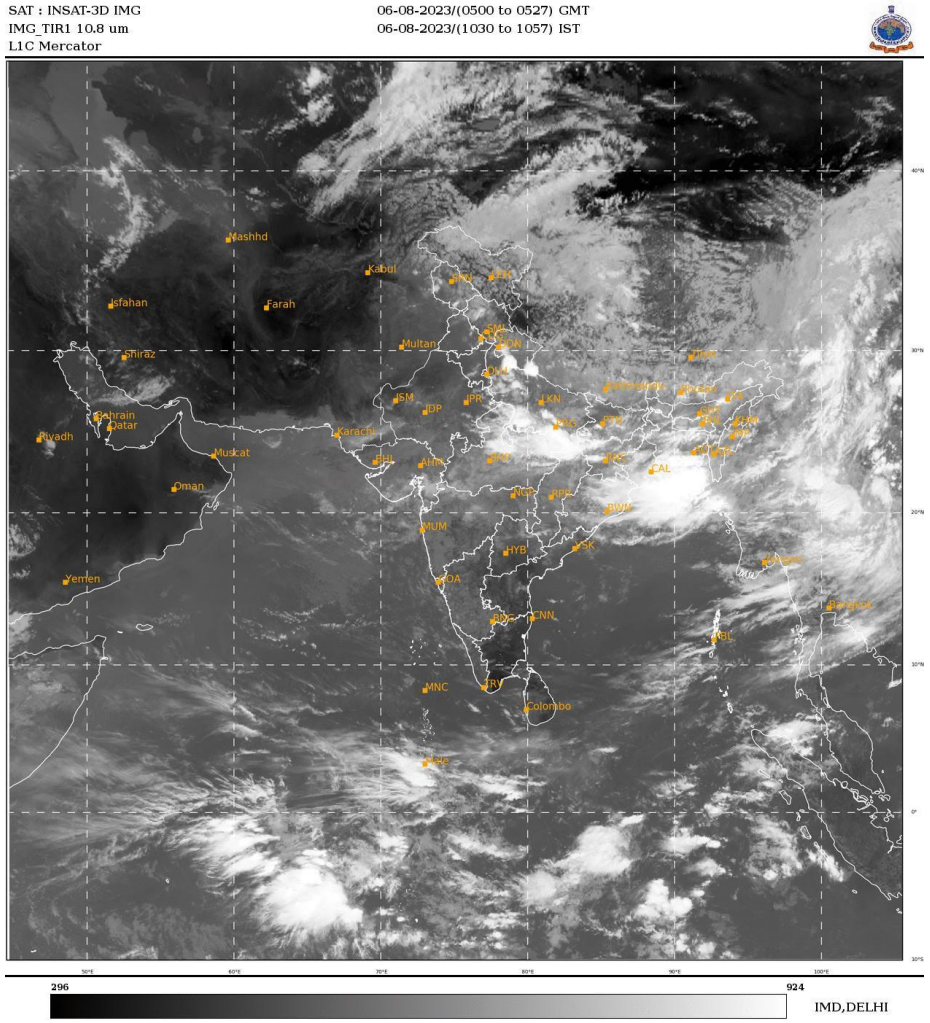
Wi-Fi Access

- Network name: FOSS4GAsia_5G
- Password: FOSS4GAsia

Geospatial Satellite Observations Datacube for Scientific Research

Session 1:

Analysis Ready Data (ARD)



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI Research Center

Analysis Ready Data for Earth Science Research

- EO data is increasingly available from various open EO archives
- Still under-utilized, full potential is yet be explored

Data utilization rate is exceeding the data exploration rate

Lack of Expertise

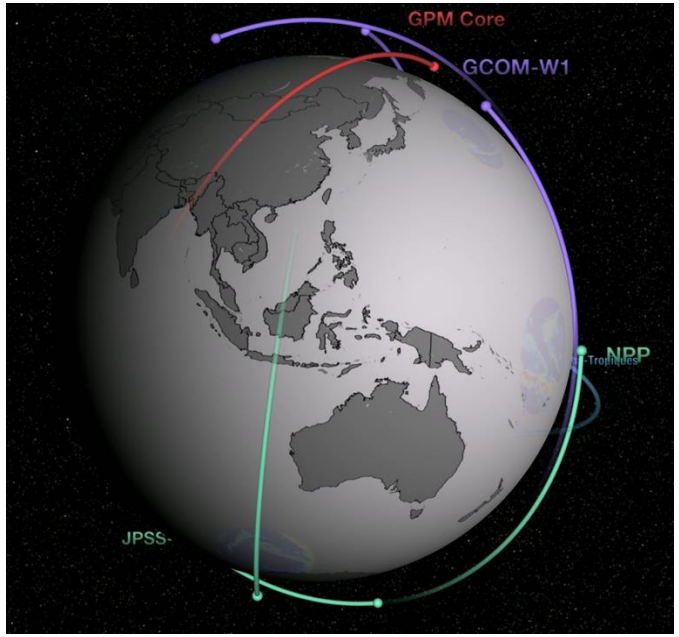
Lack of efficient infrastructure to access, process, and utilize EO data

EO data's format introduces integration challenges

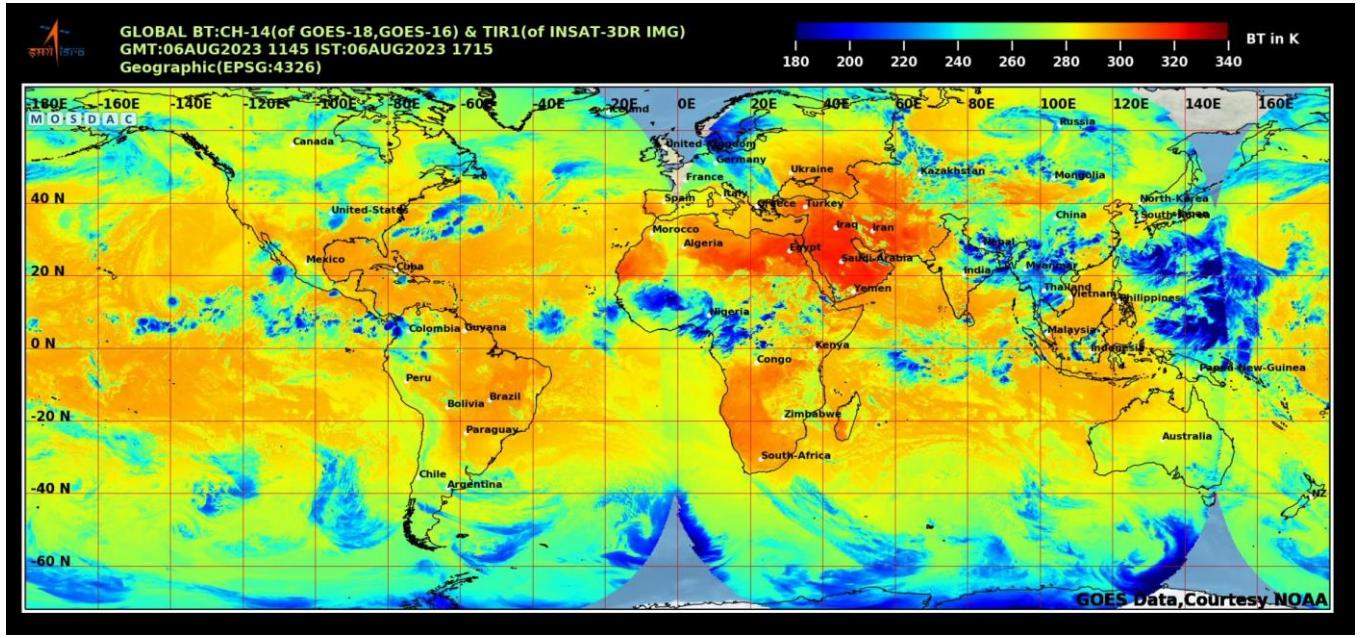
substantial effort and cost of EO data storage

Various Satellites measuring precipitation information

Low-Earth Orbit (LEO)



Geostationary Orbit (GEO)

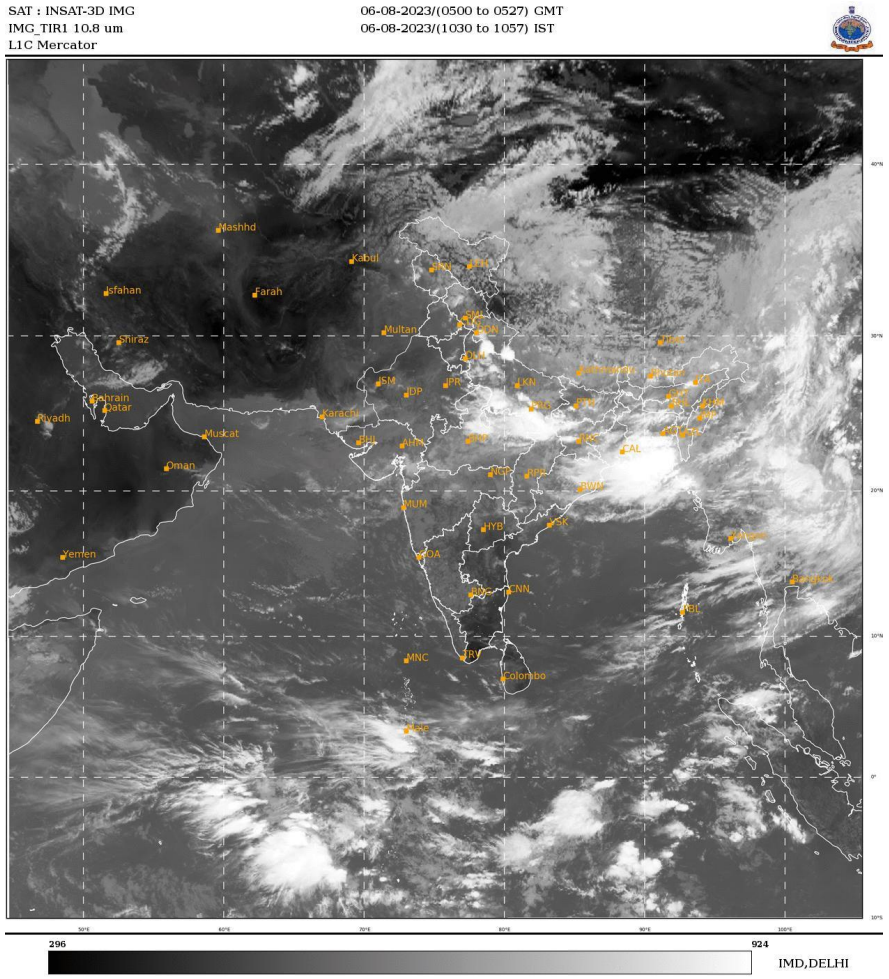


- Different sensors observe different precipitation properties
- Integrating them together and analyzing will enhance our knowledge significantly on precipitation science :
  - Aerosol-Precipitation Interaction
  - Dominant processes in convection storms
  - Detection of precipitation extremes
  - Orographic precipitation processes

Challenge: Collocation (Limitation for many researcher)

- Most LEO sensors have different revisit period;
- Spatial resolutions are different
- Different sensors capture different information: Vertical structure of cloud, Cloud-top information, Micro-physics etc
- No standard tools are available
- Need for integrating geospatial domain knowledge with Big-Data processing tools and technology

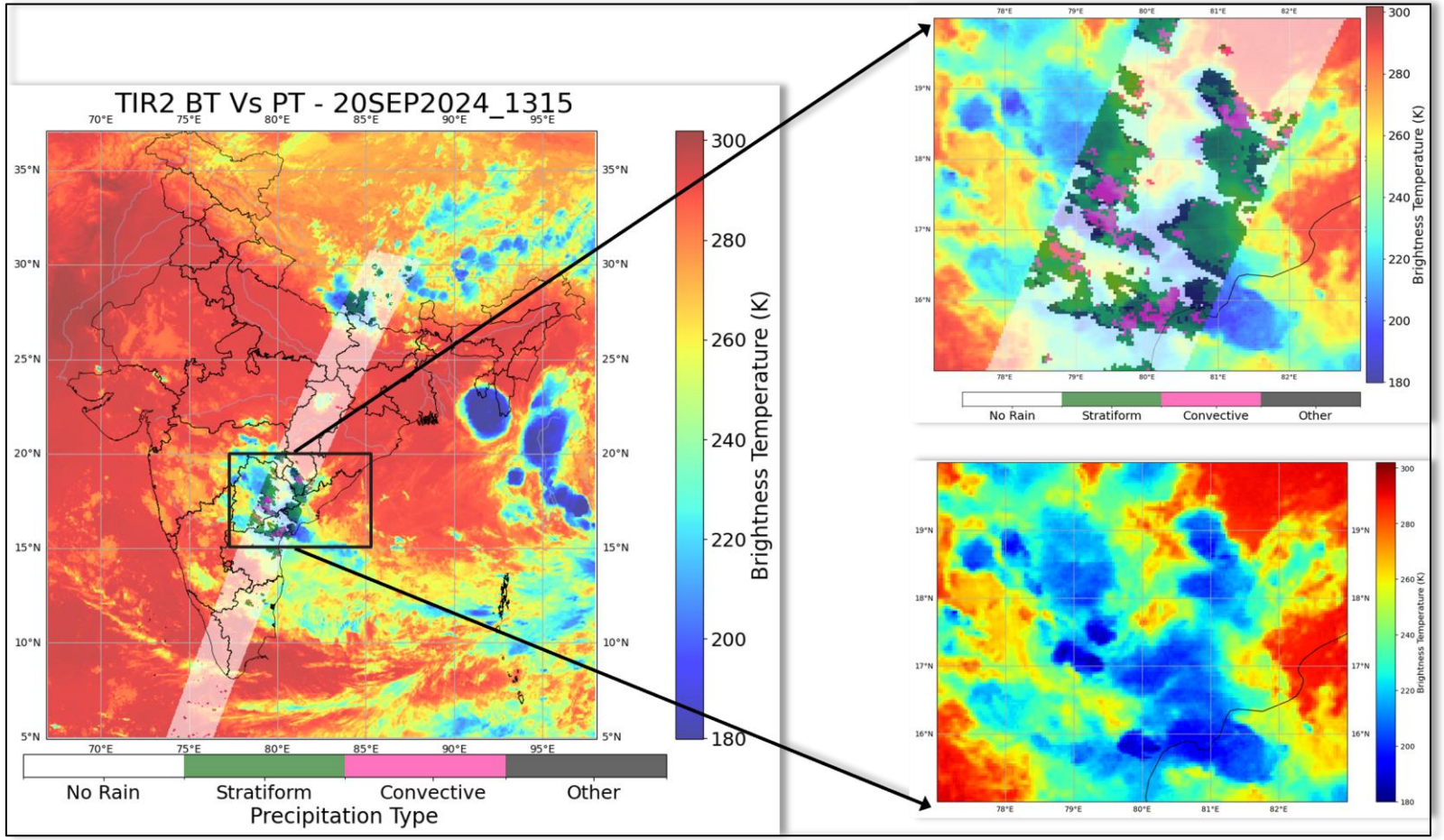
INSAT-3DR vs GPM coverage



INSAT-3DR

GPM passes

GPM-INSAT



's this data ready for precipitation related study?

Temporal File Matching

Spatial Collocation

Efficiently matching GPM pixels to the INSAT grid.

Temporal Collocation

Scan line wise temporal collocation was done with a 10-minute threshold considering the cloud system velocity.

Generate output file

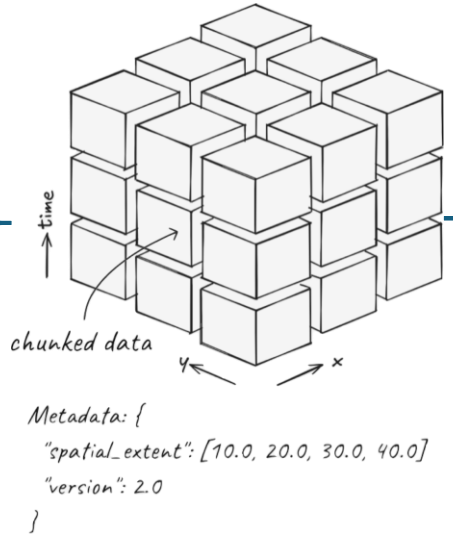
consists of two parts: the full INSAT dataset over Indian region alone and matched region containing necessary collocated GPM dats

Paradigm shift towards EO Datacube

Traditional EO workflows rely on file-based storage, manual preprocessing, and fragmented archives

Researchers face redundant, repetitive, and time-consuming preprocessing tasks such as downloading data, writing custom scripts, organizing files, and manually correcting datasets which consumes substantial time and computational effort.

As a result, researchers spend more time on data preparation rather than scientific analysis, slowing innovation and limiting productivity.



Move away from traditional local processing and data distribution methods

Lowers the barrier due to complexities related to data preparation, handling, storage, and analysis

Address EO Big Data challenges: Volume, Velocity, and Variety

Datacube

*a massive multi-dimensional array, also called “raster data” or “gridded data”; “massive” entails that we talk about sizes significantly beyond the main memory resources of the server hardware. Data values, all of the same data type, sit at grid points as defined by the **d** axes of the d-dimensional datacube. Coordinates along these axes allow addressing data values unambiguously. A d-dimensional grid is characterized by the fact that each inner grid point has exactly two neighbours along each direction; border grid points have just one. (Baumann, 2017)”*

EO Datacubes

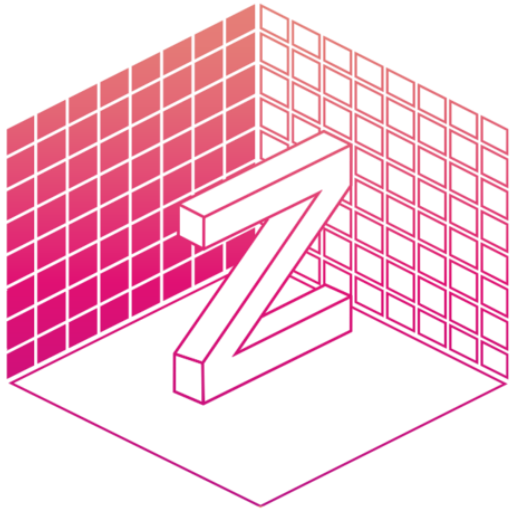
Author (Year)	Platform	Method	Key Findings / Contribution	Citation
Gorelick et al. (2017)	Google Earth Engine (GEE)	Cloud-based EO processing, multi-sensor integration	large-scale geospatial analytics, but platform is closed-source	[5]
Drivas et al. (2022)	Agriculture Data Cube (ADC), Europe	ODC-based datacube, CAP monitoring	Demonstrated scalable agriculture monitoring	[1]
Ji et al. (2025)	DeepExtremeCubes	Zarr-based storage, deep learning cloud masks, ERA5 & Sentinel inputs	Showed the effectiveness of Zarr for parallel, scalable climate-extremes modelling	[2]
Moore & Kunis (2023)	Zarr	Chunked, compressed n-dimensional array storage	Demonstrated interactive, efficient cloud access for large n-D arrays; reinforces Zarr's suitability for EO datacubes	[6]
Rasdaman GmbH (2024)	Rasdaman Array DBMS	Declarative array queries, multidimensional data operations	Powerful array database but has steep learning curve, complex setup	[9]

Fundamentals: What Is Zarr?
A Cloud-Native Format for Tensor Data

- Biggest bottleneck in the workflow can be the way we store the data

- Traditionally NetCDF, HDF5, and TIFF have been used, but as data grows bigger and more distributed, new tools are needed.

- **Zarr**:
  - a simple but powerful open-source,
  - cloud-native ***protocol***
  - for storing chunked,
  - compressed N-dimensional arrays.



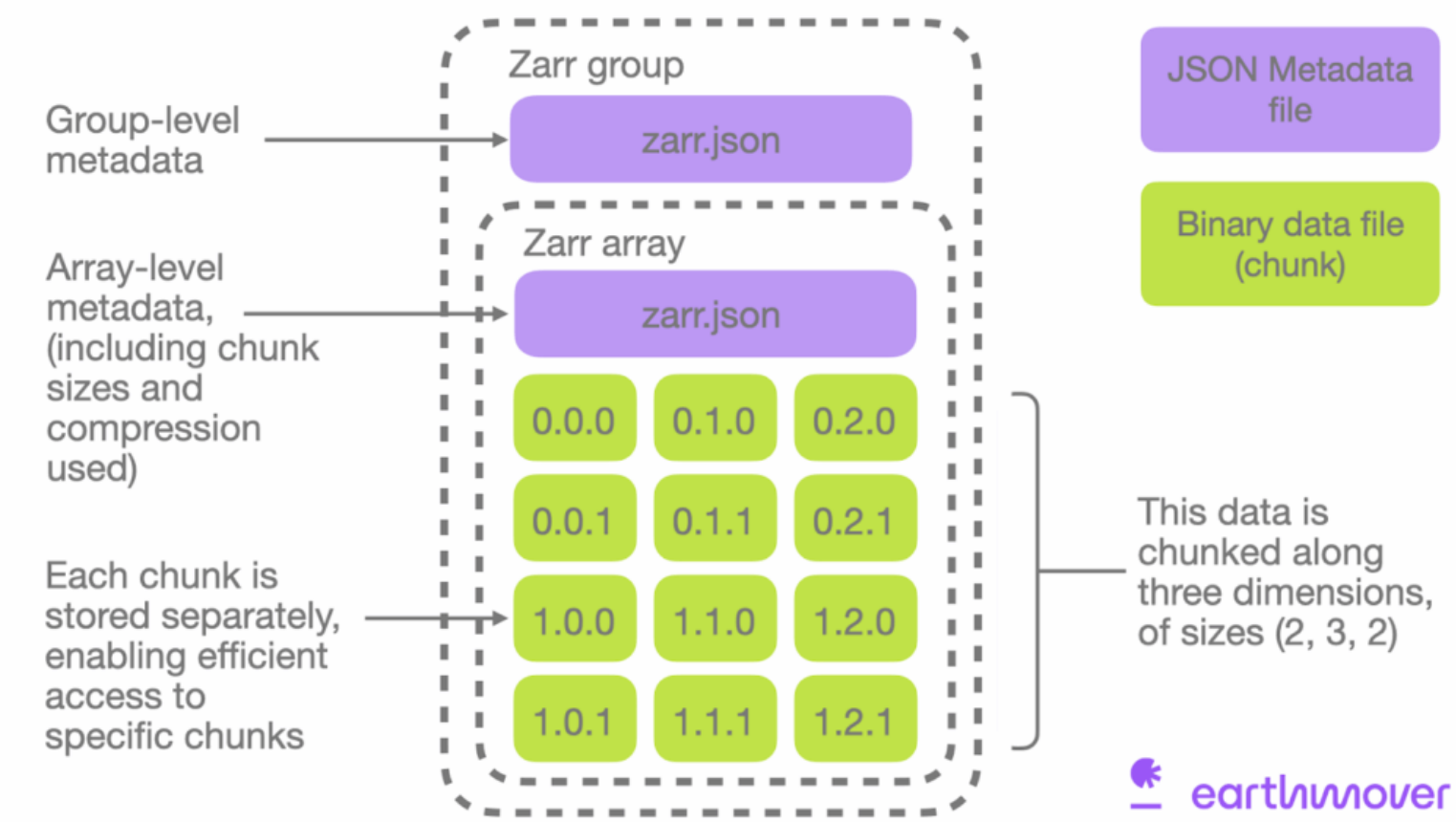
Zarr designed to be..

- **Flexible**: Works across local file systems, cloud object stores like Amazon S3 or Google Cloud Storage, and distributed file systems.
- **Efficient**: Enables fast, parallel I/O—ideal for distributed computing with tools like Dask, Ray, Spark, or Beam.
- **Self-describing**: Metadata is embedded directly alongside the data, so each array carries information about itself, like dimensions, data types, or custom attributes, without needing external documentation.
- **Open and Multi-stakeholder**: With an open spec, growing ecosystem, and community governance, Zarr is not tied to any single vendor or platform.

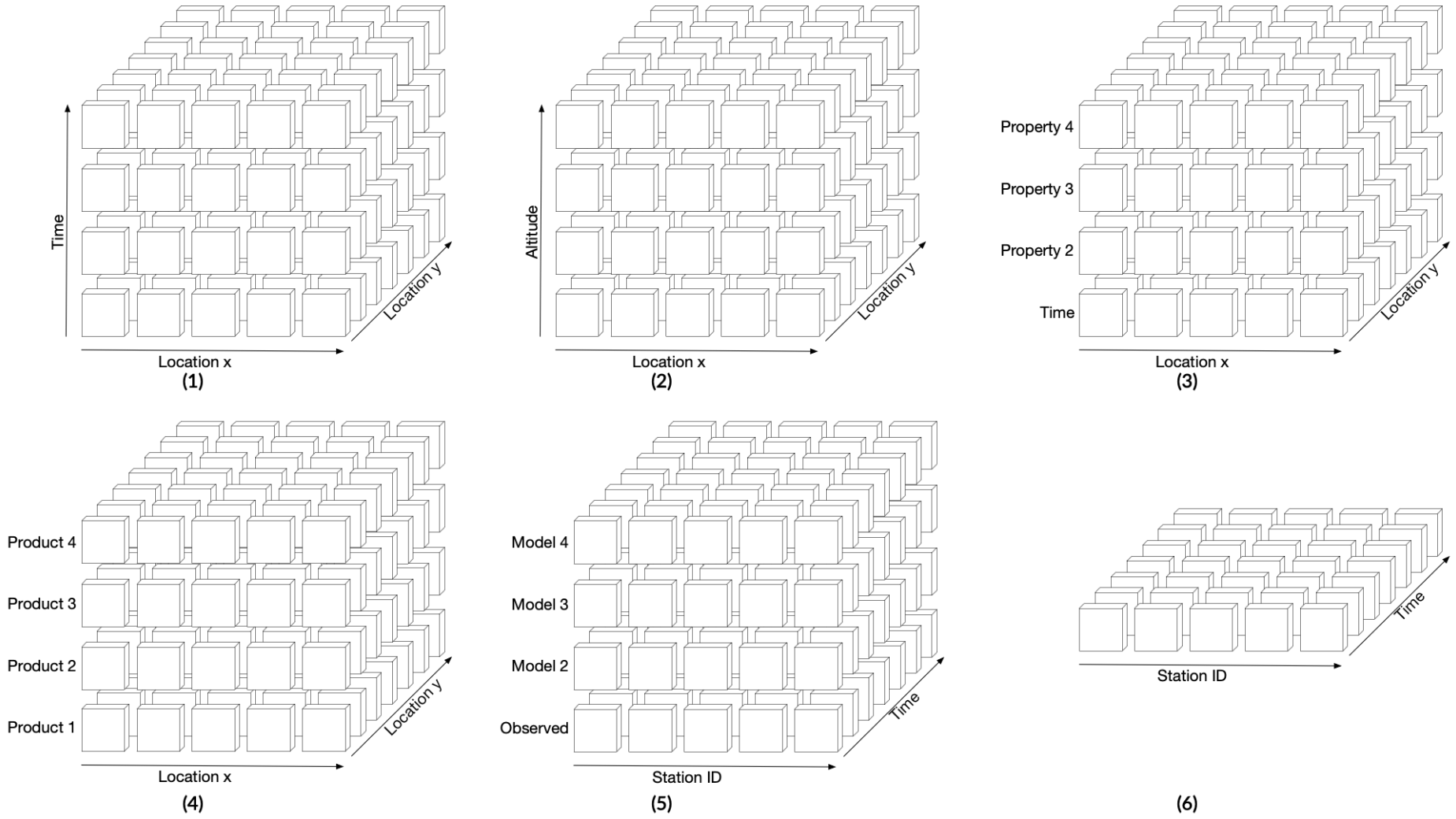
How is data stored in Zarr?

- Stores
  - Groups
    - Chunks

- Chunking
- Compression



Different implementations of data cubes



Chunked Storage for Efficient Access

Zarr stores data in chunks (small, fixed-size blocks) rather than in large monolithic files. This allows for:

- **Selective Data Loading**: Specific chunks can be loaded into the memory based (*Lazy-loading*)

- **Parallel Access**: Multiple chunks can be read or written concurrently, which is well-suited for distributed cloud environments where parallelism is a key feature.

Zarr Python API

```python
import zarr
import numpy as np

# Create a 2D Zarr array
z = zarr.create_array(
    store="data/example-1.zarr",
    shape=(100, 100),
    chunks=(10, 10),
    dtype="f4"
)

# Assign data to the array
z[:, :] = np.random.random((100, 100))
print(z.info)
```

```
Type               : Array
Zarr format        : 3
Data type          : Float32(endianness='little')
Fill value         : 0.0
Shape              : (100, 100)
Chunk shape        : (10, 10)
Order              : C
Read-only          : False
Store type         : LocalStore
Filters            : ()
Serializer         : BytesCodec(endian=<Endian.little: 'little'>)
Compressors        : (ZstdCodec(level=0, checksum=False),)
No. bytes          : 40000 (39.1K)
```

SESSION #2

A Glance at Fire Weather Index

FOREST FIRE / WILDFIRE / BUSHFIRE

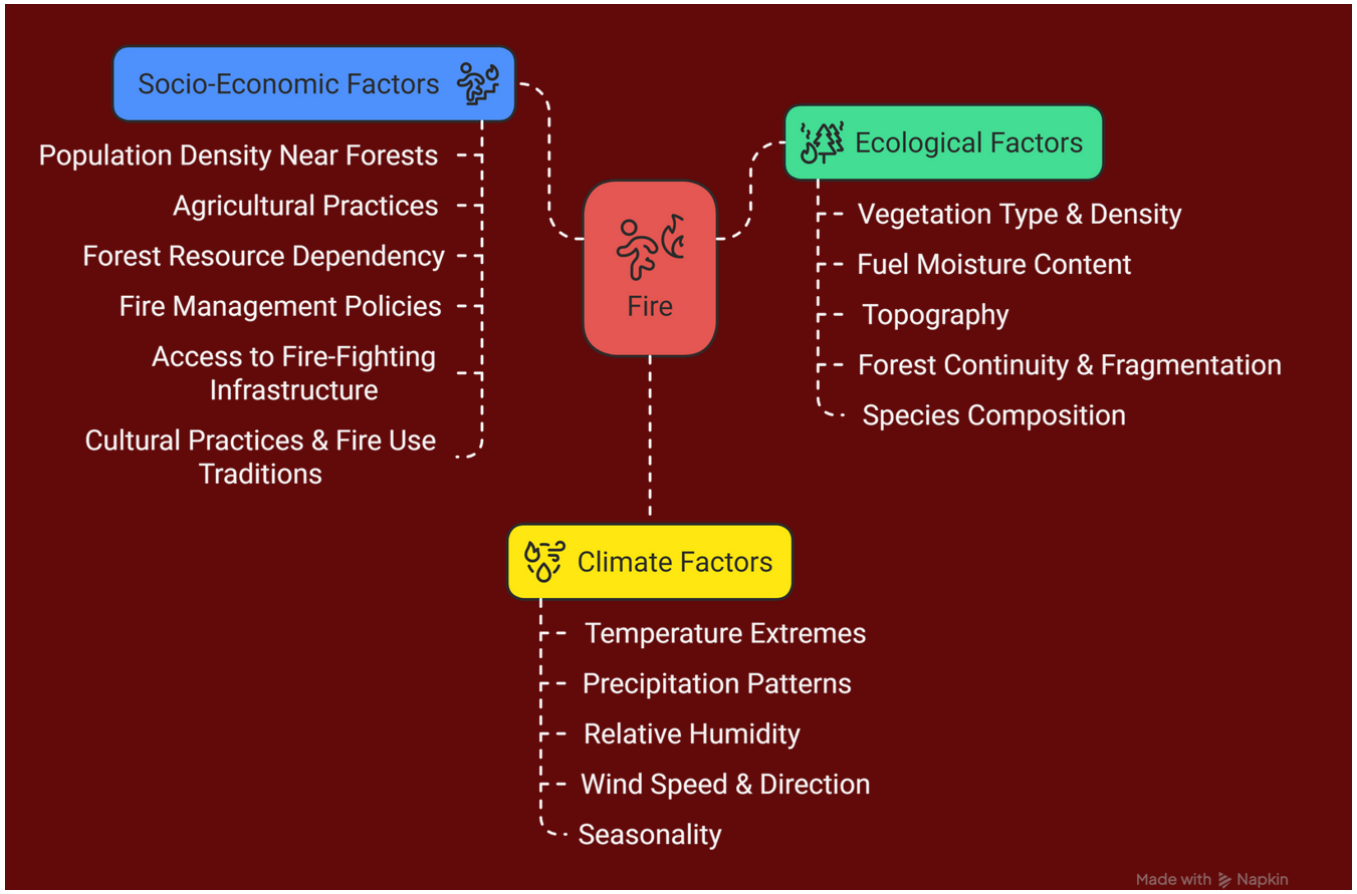
Forest fire, Uttarakhand (Feb-01 to Apr-24, 2023 & 2024)

(Source: FIRMS, NASA)

Pacific Palisades fire, Los Angeles County (Jan. 2025)

(Source: The Times, Sunday Times)

Factors Influencing Fire Occurrence and Impact

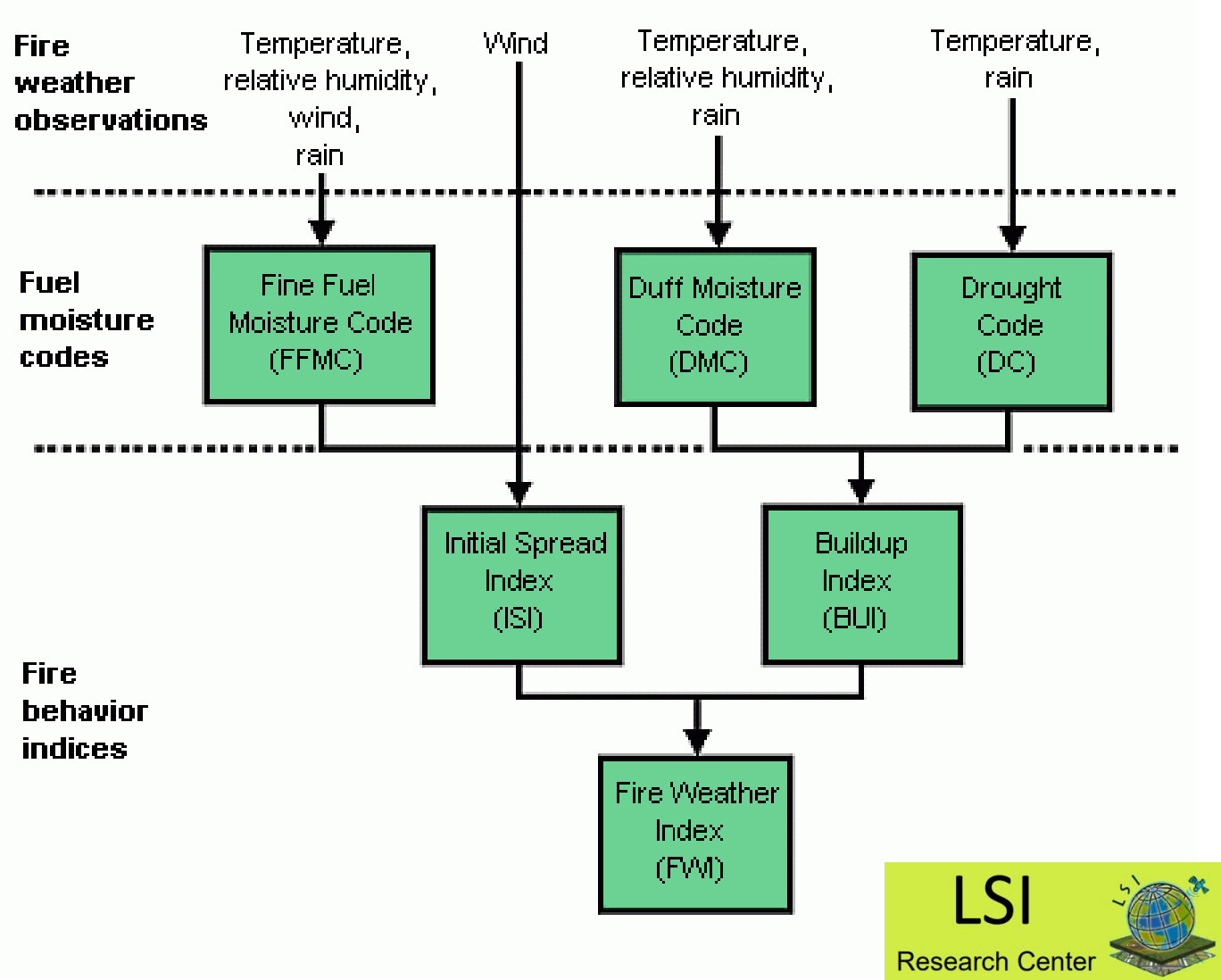


INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI Research Center

FIRE WEATHER INDEX (FWI)

- A meteorologically based index developed by the Canadian Forestry Service to estimate forest fire ignition and spread conditions
- Consists of different components that account for the effects of fuel moisture and wind on fire behaviour and spread
- A higher index value indicates better weather conditions for wildfires and so, a higher risk
- Helps shape long-term investments under a changing



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI Research Center

Dataset

FWI can be calculated using climatic data from historical simulations provided by **ECMWF ERA5 reanalysis** datasets

Climate Data Store Datasets Applications User guide Forum Live

ERA5 hourly data on single levels from 1940 to present

Overview Download Documentation

Info

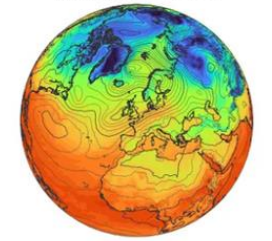
11 Dec 2025

Please note that a dedicated catalogue entry for this dataset, post-processed and stored in Analysis Ready Cloud Optimized (ARCO) format (Zarr), is available for optimised time-series retrievals (i.e. for retrieving data from selected variables for a single point over an extended period of time in an efficient way). You can discover it here

ERA5 is the fifth generation ECMWF reanalysis for the global climate and weather for the past 8 decades. Data is available from 1940 onwards. ERA5 replaces the ERA-Interim reanalysis.

Reanalysis combines model data with observations from across the world into a globally complete and consistent dataset using the laws of physics. This principle, called data assimilation, is based on the method used by numerical weather prediction centres, where every so many hours (12 hours at ECMWF) a previous forecast is combined with newly available observations in an optimal way to produce a new best estimate of the state of the atmosphere, called analysis, from which an updated, improved forecast is issued.

ERA5 2 metre temperature and Mean sea level pressure
1 January 2023 at 00:00 UTC



-40 -30 -20 -10 0 10 20 30 40
2 metre temperature (°C)

Step 1 – Download Meteorological Data (ERA5 from Copernicus)

1. Login to the **Copernicus Climate Data Store (CDS)**.
2. Select dataset:
   - **ERA5 hourly data on single levels from 1940 to present**.
   - Product Type: **Reanalysis**.
3. Select required **variables**:
   - 10m u-component of wind (`u10`).
   - 10m v-component of wind (`v10`).
   - 2m dewpoint temperature (`d2m`).
   - 2m temperature (`t2m`).
   - Total precipitation (`tp`).
4. Select **temporal coverage**: Specific year(s), month(s), day(s), and hour(s).
5. Define **geographical area** (subregion extraction):
   - N=37, W=68, S=6, E=98.
6. Choose **output format**: **NetCDF**.
7. Submit request and download files.

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI Research Center

Session 3

Demonstration - Live

Agenda

1. Scope of learning
2. Installation instructions
3. Code walk-through
    1. Windows – zarr version 3
    2. Linux – zarr version 2
4. Local host experiments
    1. Windows Client to Windows Server
    2. Linux Client to Linux Server
5. Dedicated client-server systems experiments
    1. Windows Client to Linux Server
    2. Linux Client to Windows Server
6. Other Case Studies & Queries
7. Visualization ideas

Learning objectives

- OBJ #1: Setting up python (conda) environment
- OBJ #2: Generation of datacube from publicly available dataset
- OBJ #3: Setting up pygeoapi and ingesting fire weather index ARD data
- OBJ #4: Running spatio-temporal OLAP queries and analysis

Session 4

Hands-On Experiments

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI
Research Center

Anaconda Installation

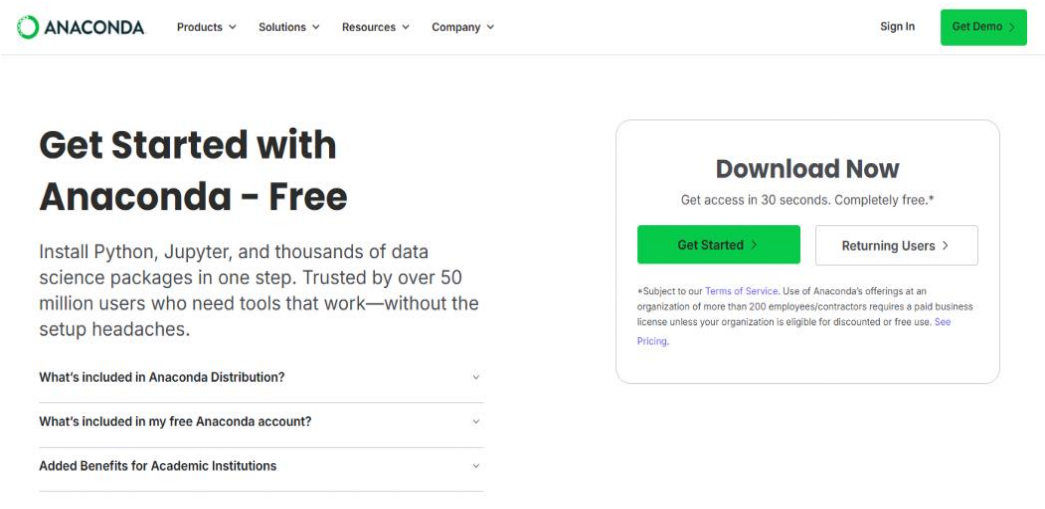
For Windows

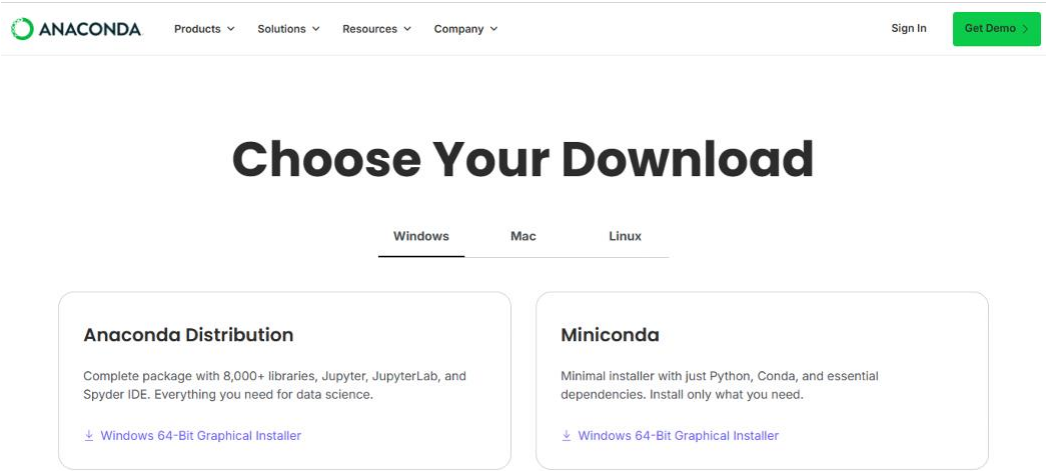
Installation Instructions

- Download Anaconda Graphical Installer for your Operating System:

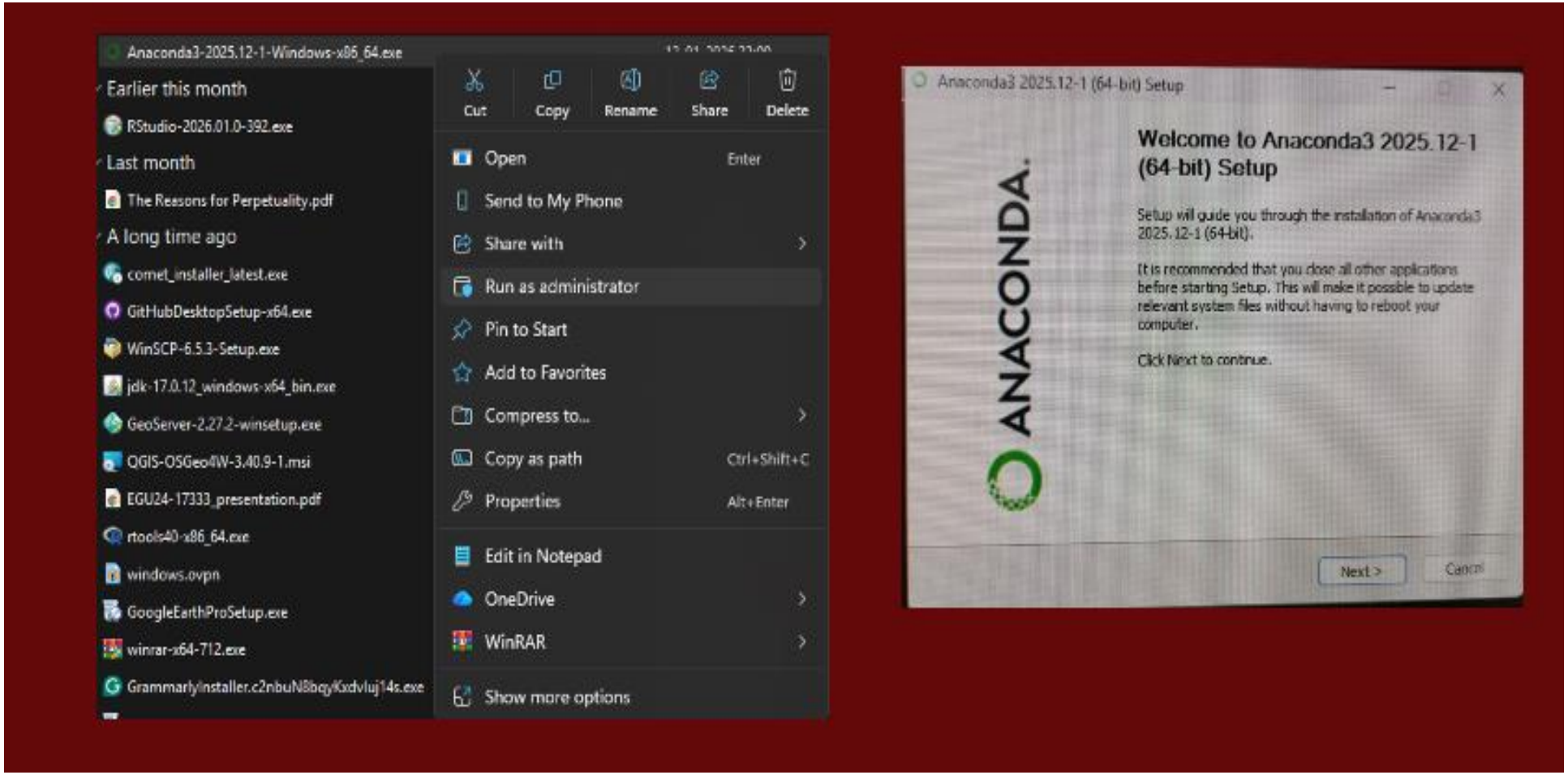
https://www.anaconda.com/download/

- Choose Anaconda Distribution (only)
- Google Sign In is supported
- Requires about 1-1.5 GB free space
- Install in C drive (Windows); /home/user (Linux)

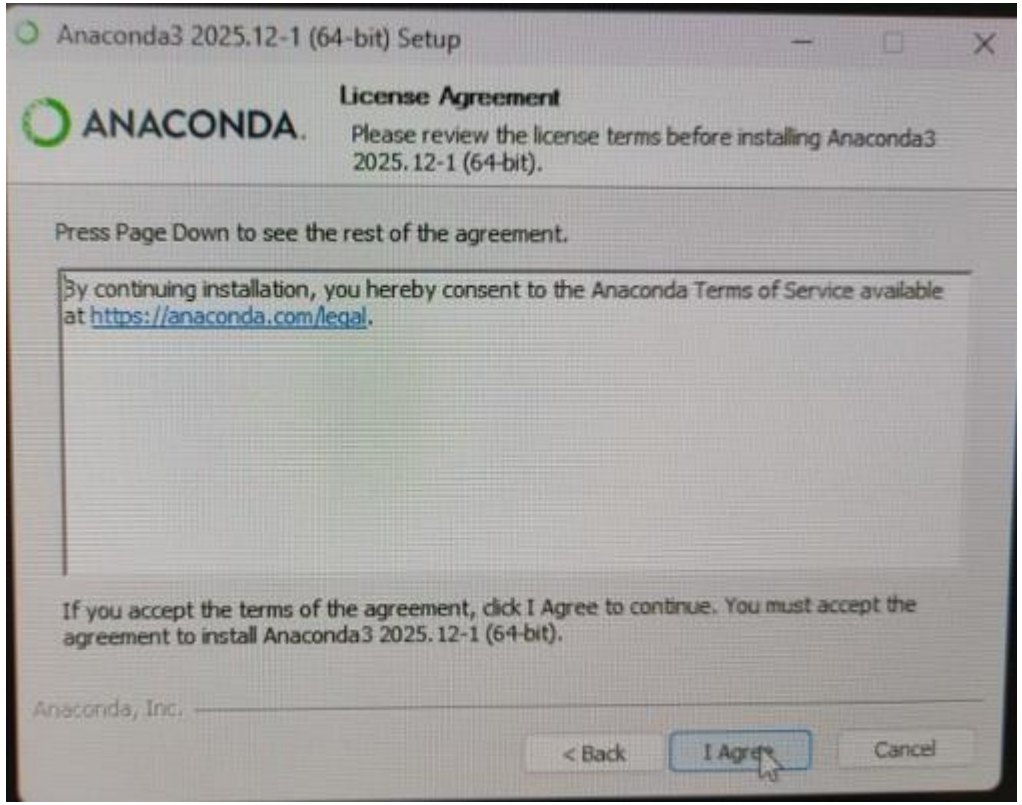


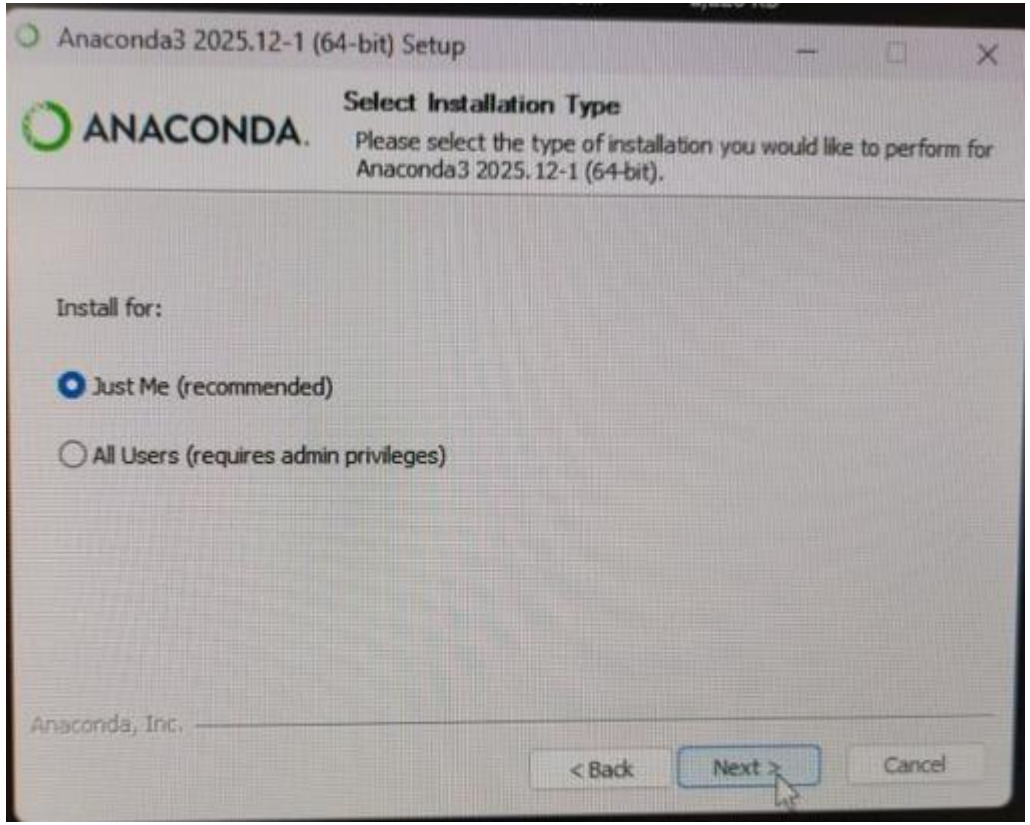


Anaconda installer

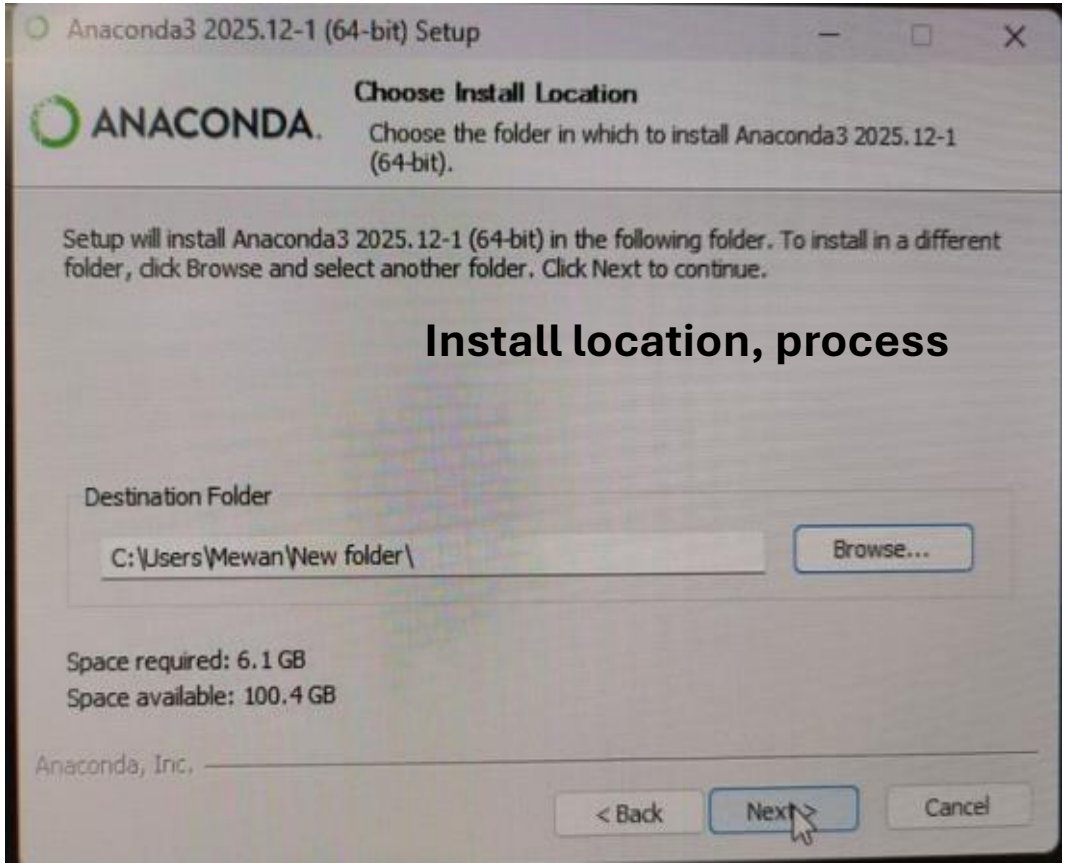


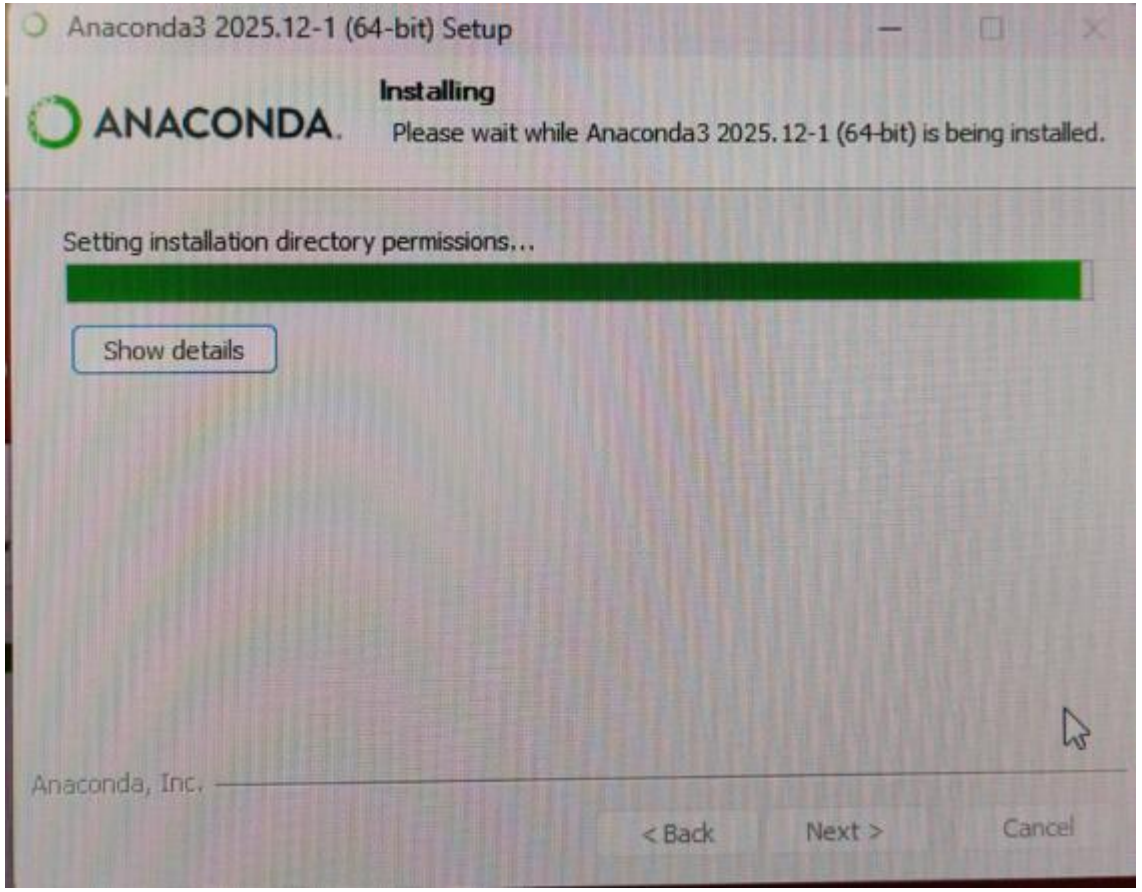
License, installation Type (Users)



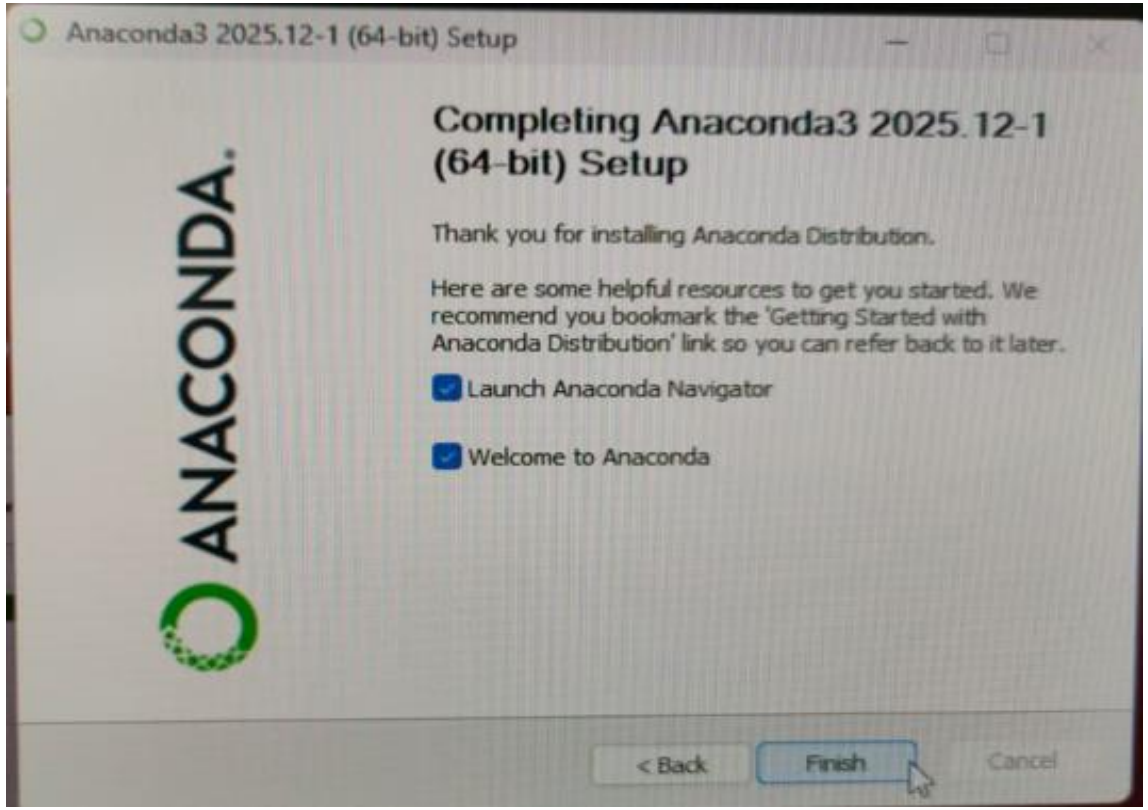


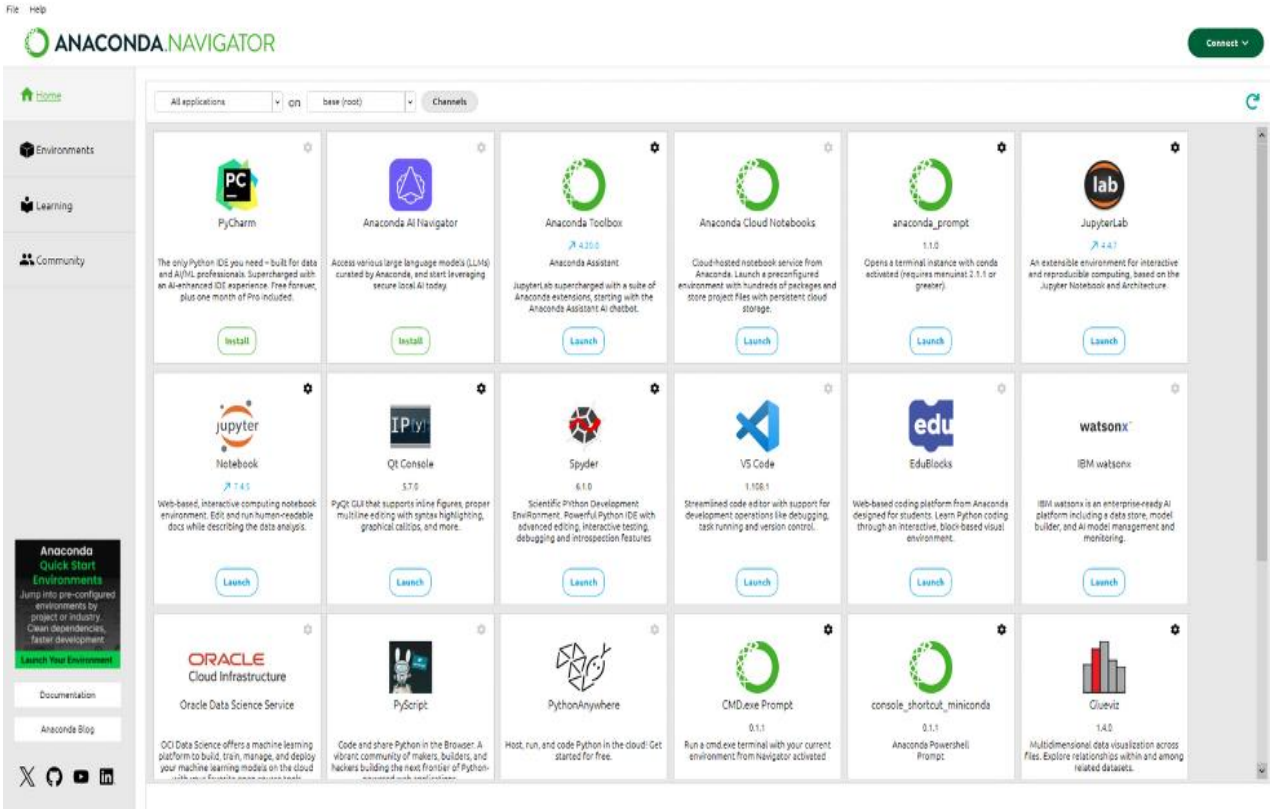
Install location, process





Success #1!


Anaconda3 2025.12-1 (64-bit) Setup
ANACONDA.
Completing Anaconda3 2025.12-1 (64-bit) Setup
Thank you for installing Anaconda Distribution.
Here are some helpful resources to get you started. We recommend you bookmark the 'Getting Started with Anaconda Distribution' link so you can refer back to it later.
Launch Anaconda Navigator
Welcome to Anaconda
< Back
Finish
Cancel


File Help
ANACONDA.NAVIGATOR
Connect
Home
Environments
Learning
Community
All applications
on
base (root)
Channels
PyCharm
Anaconda AI Navigator
Anaconda Toolbox
Anaconda Cloud Notebooks
anaconda_prompt
JupyterLab
Notebook
Qt Console
Spyder
VS Code
EduBlocks
IBM watsonx
Oracle Data Science Service
PyScript
PythonAnywhere
CMD.exe Prompt
console_shortcut_miniconda
Glueviz
Install
Launch
Anaconda Quick Start Environments
Jump into pre-configured environments by project or industry. Clean dependencies, faster development.
Launch Your Environment
Documentation
Anaconda Blog

Folder Creations

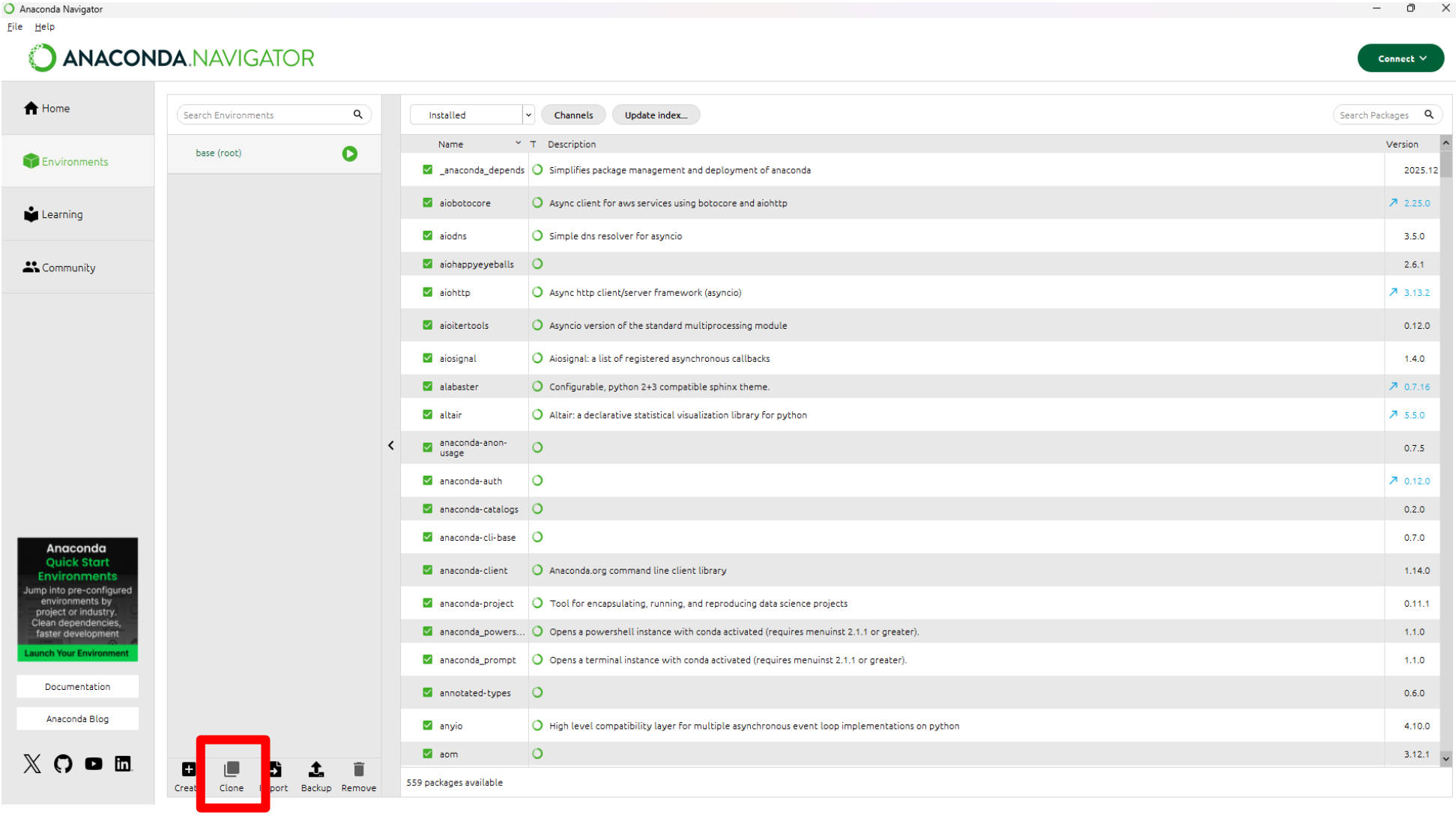
- Create 2 folders
- C://zarr - Fire Weather Index geodatacube will get created within this folder
- C://Users//<User_Name>/Test - This Test folder is where all code and data files will be stored

- Place all the unzipped files in Test folder
  - INSAT_GEO.yml - Main environment to be created in Anaconda

  - India shape file setup (7 files) with prefix: India_Country_Boundary
  - Fire Weather Index Netcdf file - FWI_2014_2018_combined.nc
  - pygeoapi-config.yml - Configuration File for Server
  - openapi.yaml - Support configuration file for server

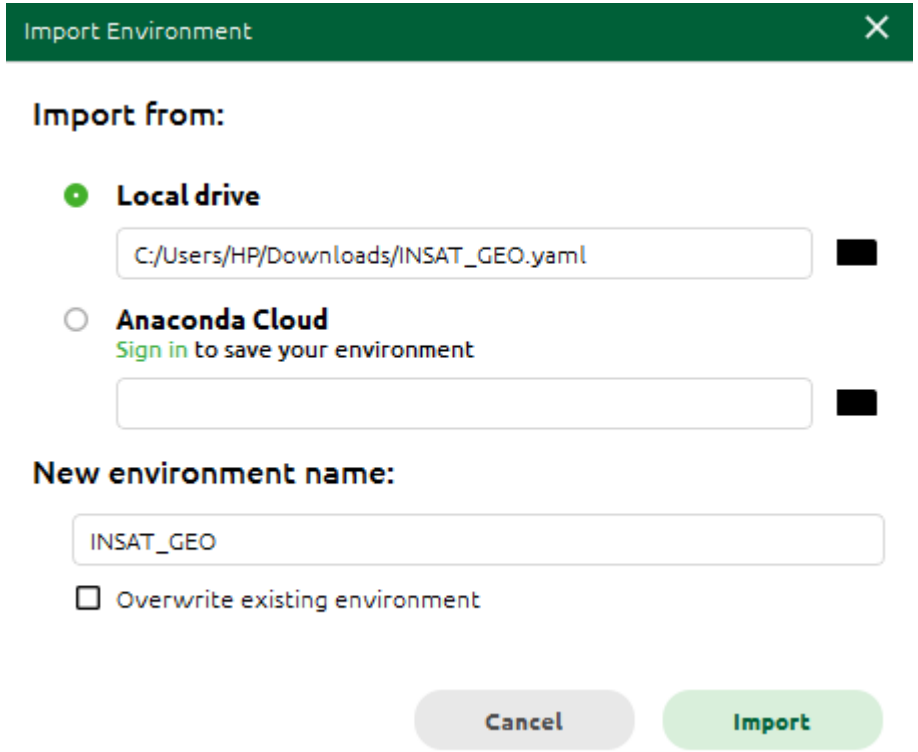
- Never double click any file to open
- Always use File Menu -> Open option to prevent automatic renaming with numbers suffixed
- Don’t copy paste file names - It can introduce trailing, hidden empty spaces

Environment Creation - Import yml



Steps to create working environment

- Import INSAT_GEO.yml from local drive (C drive – Test Folder)
- Retain the same name for the new environment
- It will take up to 15 minutes to auto install all required libraries to setup the environment
- Press yes/y whenever prompted by Conda to install all libraries



File Help

ANACONDA.NAVIGATOR

Connect

Home

Environments

Learning

Community

Search Environments

base (root)

INSAT_GEO

Installed | Channels | Update index... | Search Packages

Name	T	Description	Version
_openmp_mutex		Openmp implementation mutex	4.5
affine		Matrices describing affine transformation of the plane.	2.4.0
annotated-types			0.7.0
anyio		High level compatibility layer for multiple asynchronous event loop implementations on python	4.12.0
attrs		Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods).	25.4.0
aws-c-auth			0.9.3
aws-c-cal			0.9.13
aws-c-common		Core c99 package for aws sdk for c. includes cross-platform primitives, configuration, data structures, and error handling.	0.12.6
aws-c-compression			0.3.1
aws-c-event-stream		C99 implementation of the vnd.amazon.eventstream content-type.	0.5.7
aws-c-http			0.10.7
aws-c-io			0.23.3
aws-c-mqtt			0.13.3
aws-c-s3			0.11.3
aws-c-sdkutils			0.2.4
aws-checksums		Cross-platform hw accelerated crc32c and crc32 with fallback to efficient sw implementations. c interface with language bindings for each of our sdks.	↗ 0.2.7
aws-crt-cpp			0.35.4
aws-sdk-cpp		C++ library that makes it easy to integrate c++ applications with aws services	1.11.606
babel		Utilities to internationalize and localize python applications	2.17.0
backports.zstd			1.3.0

268 packages available

Create Clone Import Backup Remove

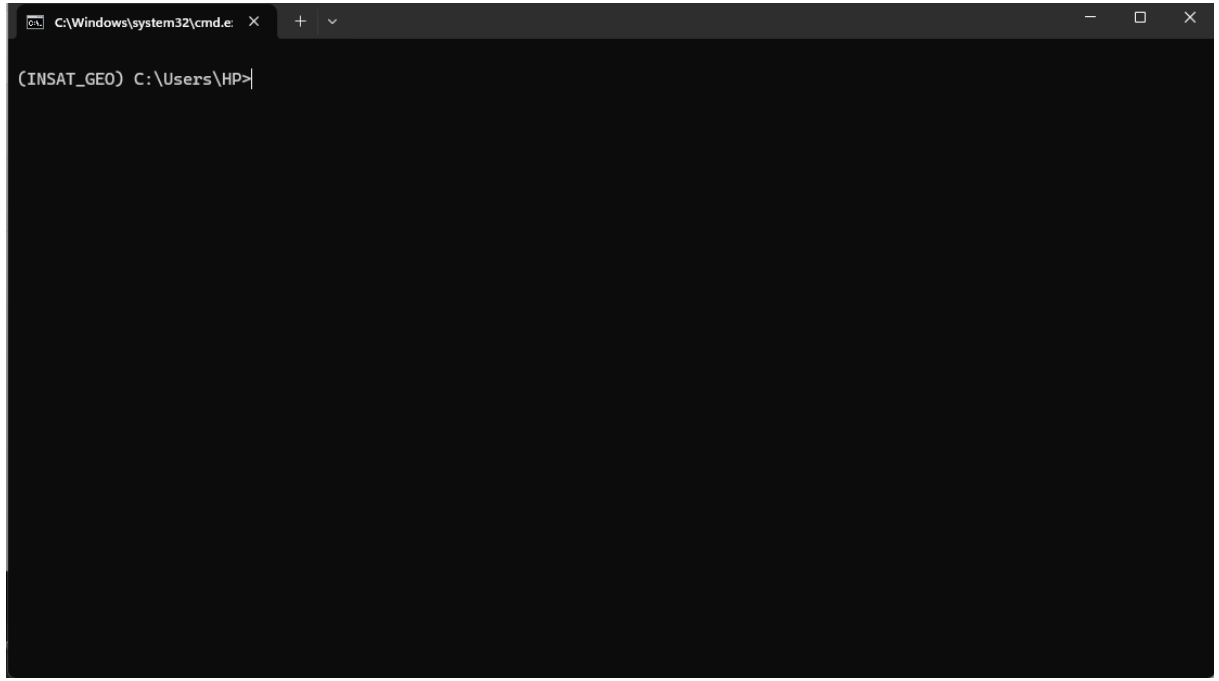
Anaconda
Quick Start Environments
Jump into pre-configured environments by project or industry. Clean dependencies, faster development
Launch Your Environment

Documentation

Anaconda Blog

Starting terminal...

- Left click that green icon beside INSAT_GEO label
- It will load the environment for use in few seconds
- Right click now and choose "Open Terminal"



Warning!

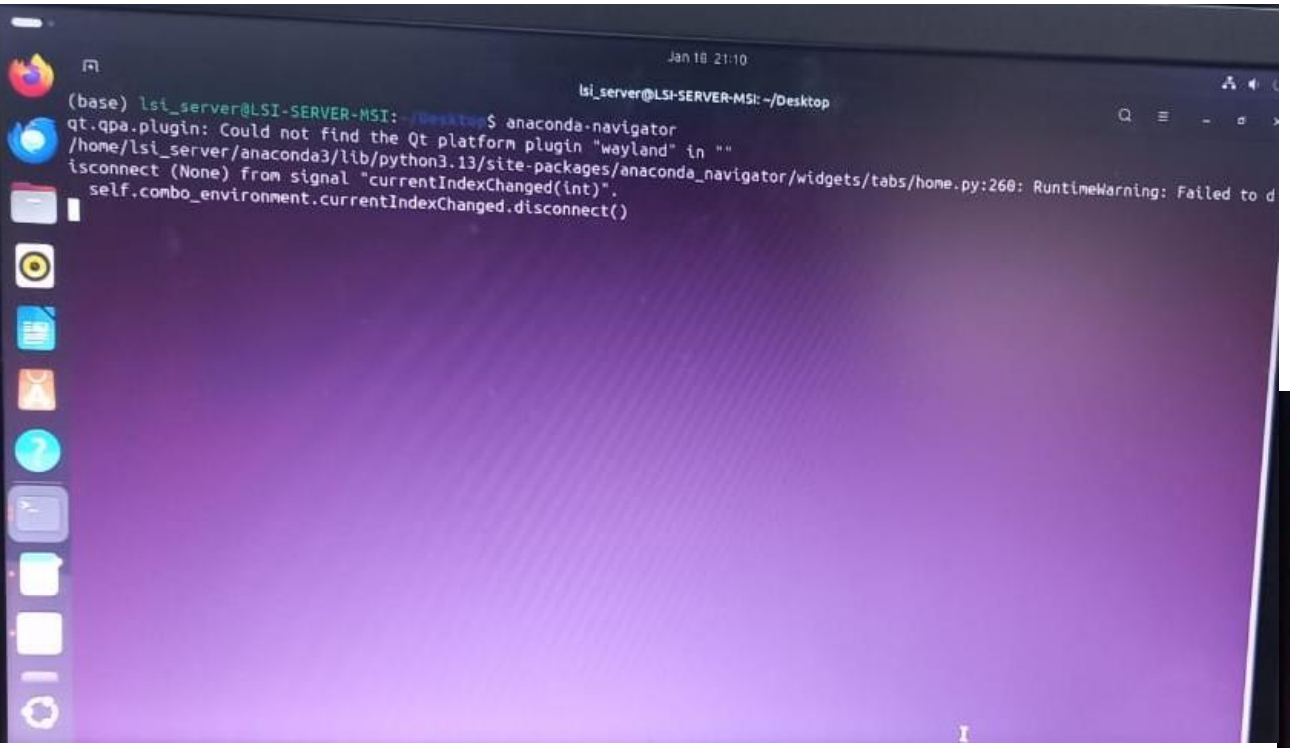
- Do not use the Windows Service Layer – Ubuntu OS as client or server in the same laptop

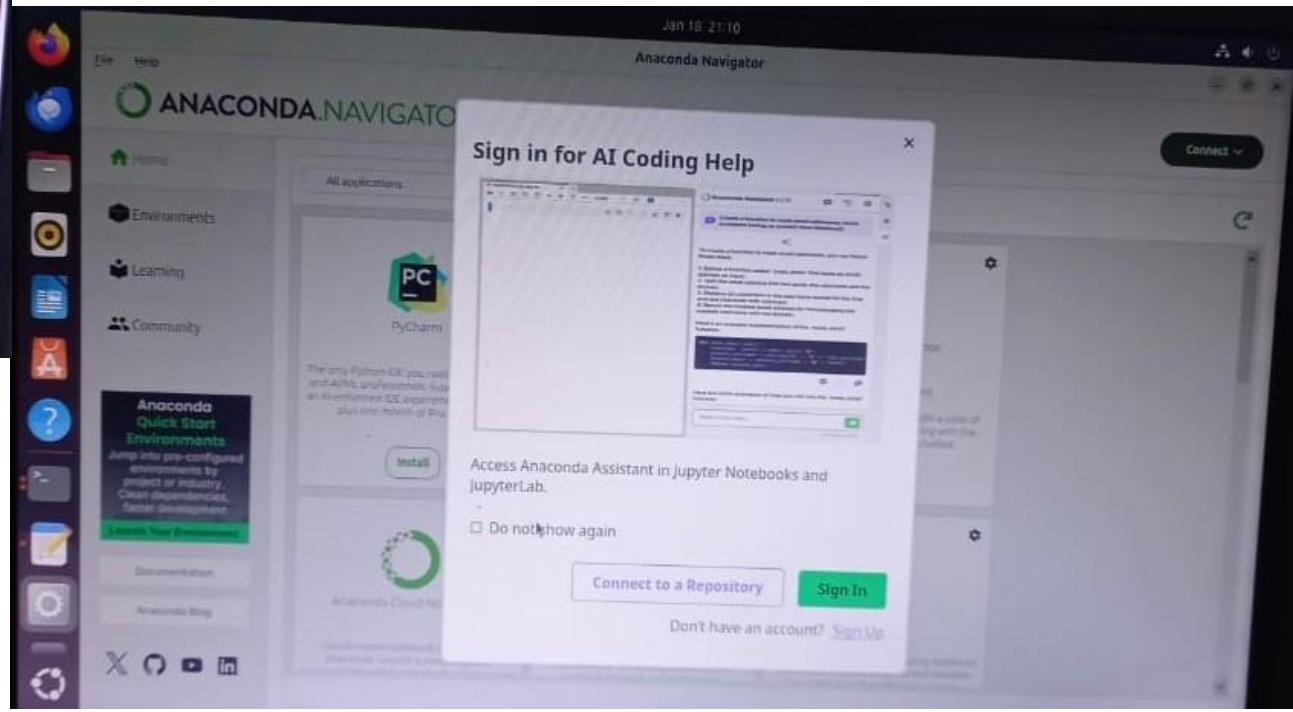
Anaconda installation

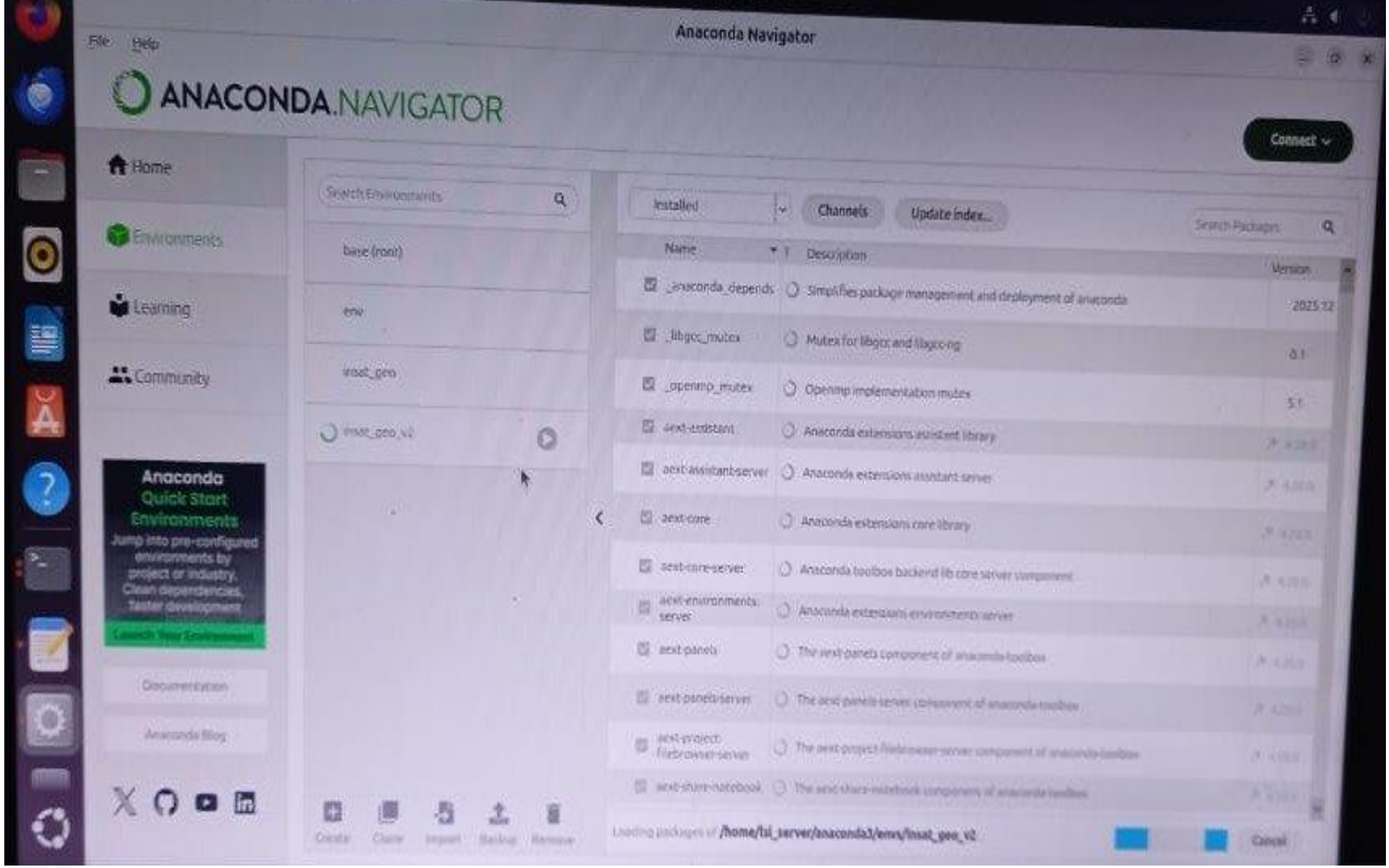
For Linux

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

LSI
Research Center


Jan 18 21:10
lsi_server@LSI-SERVER-MSI: ~/Desktop
(base) lsi_server@LSI-SERVER-MSI:
$ anaconda-navigator
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in ""
/home/lsi_server/anaconda3/lib/python3.13/site-packages/anaconda_navigator/widgets/tabs/home.py:260: RuntimeWarning: Failed to d
isconnect (None) from signal "currentIndexChanged(int)".
self.combo_environment.currentIndexChanged.disconnect()


Jan 18 21:10
Anaconda Navigator
File
Help
ANACONDA.NAVIGATO
Connect
Environments
Learning
Community
PyCharm
Install
Anaconda
Quick Start
Environments
Sign in for AI Coding Help
Access Anaconda Assistant in Jupyter Notebooks and JupyterLab.
Do not show again
Connect to a Repository
Sign In
Don't have an account? Sign Up


Anaconda Navigator
File
Help
ANACONDA.NAVIGATOR
Connect
Home
Environments
Learning
Community
Search Environments
base (root)
env
insat_geo
insat_geo_v2
Installed
Channels
Update Index...
Search Packages
Name
Description
Version
_anaconda_depends
Simplifies package management and deployment of anaconda
2025.12
_libgcc_mutex
Mutex for libgcc and libgcc-ng
0.1
_openmp_mutex
Openmp implementation mutex
5.1
Anaconda extensions assistant library
Anaconda extensions assistant server
Anaconda extensions core library
Anaconda toolbox backend lib core server component
Anaconda extensions environments server
The anaconda-toolbox
The anaconda-toolbox
The anaconda-toolbox
Anaconda
Quick Start
Environments
Jump into pre-configured environments by project or industry. Clean dependencies, faster development.
Documentation
Anaconda Blog
Create
Clone
Import
Backup
Remove
/home/tsi_server/anaconda3/envs/insat_geo_v2
Cancel

Few basic commands for handy reference!

- pwd- will show the full path of the present working directory (folder)
- ls/dir – will list all the contents in the current directory
- cls/clear – clears the terminal clear of all commands
- cd - Change directory
- cat session_log.txt - The file will contain everything exactly as it appeared on your screen, including the command prompts and any error messages
- exit – to exit the session
- Ctrl + C - to gracefully terminate a running process
- mkdir - make directory
- gvim/vim - text editor in Linux

Installation Instructions

- chmod Anaconda3-2025.12-1-Linux-x86-64.sh
- To install:
  - Prompt: >> ./Anaconda3-2025.12-1-Linux-x86-64.sh
- >> anaconda-navigator

Environment Creation

- conda create –n insat-geo-v2 python=3.10
- conda activate insat-geo-v2 python=3.10

- Conda env create –f insat-geo-v2

- Create a directory called zarr: mkdir –p /home/lsi_server/zarr
  - Check **“your”** system path in Home folder using pwd command

- For various imports:
  - conda install *dask zarr geopandas xarray pandas numpy geopandas numpy pyproj libnetcdf rioxarray rasterio*
  - NOTE: insat-geo-v2.yml also provided

More instructions for Linux users

- conda install –c conda-forge zstandard
- sudo apt install net-tools
- export ZARR_V3_EXPERIMENTAL_API=1
- chmod –R 777 /home/lsi_server/zarr/fwi_india_2016.zarr

Code walk-through

- Server-side activities
  - Script 1 → Preparing ARD in Zarr format (NC_ZARR_Win.py / NC_ZARR_Lin.py)
  - Script 2 → Publishing Zarr data over pygeoapi (pygeoapi-config.yml)

- Client-side activities
  - Script 3 → Consuming data as slices (Win_Client_Query.py, Lin_Client_Query.py)

Important note

- Code walk-through is for Windows version of all 3 scripts
- For Linux, only the input and output path contents change
- No other difference – Users must take note and make required changes

Code walk-through – Server on Windows

1. Generating ARD Ready Datasets in Zarr format (NC_ZARR_Win.py / NC_ZARR_Lin.py)
2. Pushing it to Server (pygeoapi.yaml)

Generating Zarr of ARD

- Python script takes Fire Weather Index (FWI) data in NetCDF (nc) format
  - Clips it to spatial extent of India
  - Fixes time and spatial metadata
  - Converts it into a cloud-optimized Zarr v3 data cube
    - ready for pygeoapi (EDR / Coverage services)

Library imports

```python
# -*- coding: utf-8 -*-

# ==================================================
# PROJ SETUP (MUST BE FIRST)
# ==================================================
import os
import pyproj
from pyproj import CRS

os.environ["PROJ_LIB"]  = r"C:\Users\Lenovo\anaconda3\envs\INSAT_GEO\Library\share\proj"
os.environ["PROJ_DATA"] = r"C:\Users\Lenovo\anaconda3\envs\INSAT_GEO\Library\share\proj"

print("PROJ version:", pyproj.proj_version_str)
print("PROJ data dir:", pyproj.datadir.get_data_dir())
print(CRS.from_epsg(4326))
print("\n**************\n")
```

- Configures **PROJ**, the library that handles **coordinate reference systems (CRS)**.
- Ensures Python knows **where the CRS definitions live** on Windows.
- On Windows + Conda, PROJ paths are **not always detected automatically**.
  - If PROJ is misconfigured:
    - CRS assignment fails
    - Clipping fails
    - pygeoapi may silently misinterpret coordinates
- “We are explicitly confirming that EPSG:4326 (latitude–longitude, WGS84) is available and working.”

Need specific imports

```
# ==================================================
# IMPORTS
# ==================================================
import xarray as xr
import geopandas as gpd
import rioxarray  # noqa
import zarr
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Library	Role
xarray	Multi-dimensional scientific data (NetCDF, Zarr)
geopandas	Vector geometries (India boundary)
rioxarray	Spatial metadata + clipping
zarr	Cloud-optimized storage
numpy / pandas	Time and array handling
matplotlib	Quick sanity visualization

Loading and reading inputs

```python
# ==================================================
# PATHS
# ==================================================
nc_path  = r"C:\Users\Lenovo\Test\fwi_output_2016.nc"
shp_path = r"C:\Users\Lenovo\Test\India_Country_Boundary.shp"
zarr_out = r"C:\zarr\fwi_india_2016.zarr"

# ==================================================
# LOAD NETCDF
# ==================================================
print("Opening FWI NetCDF...")

ds = xr.open_dataset(
    nc_path,
    decode_times=True,
    use_cftime=False
)

```

- Opens NetCDF **lazily** (data not fully loaded into memory).
- Attempts to decode time into real dates.
- “xarray treats NetCDF as labeled data, not just raw arrays”
- India shape file is read for boundary
- Output zarr is to be saved in C drive zarr folder

Time fix – a critical module

```python
# ==================================================
# TIME FIX (CRITICAL)
# ==================================================
if not np.issubdtype(ds.time.dtype, np.datetime64):
    print("Fixing integer time axis ? datetime64")

    # Try CF decoding manually
    ds = xr.decode_cf(ds, use_cftime=False)

# Final guarantee
if not np.issubdtype(ds.time.dtype, np.datetime64):
    print("Forcing daily datetime index as fallback")

    time_index = pd.date_range(
        start="2016-01-01",
        periods=ds.sizes["time"],
        freq="D"
    )

    ds = ds.assign_coords(time=("time", time_index))

print("Final time dtype:", ds.time.dtype)


print("Original variables:", list(ds.data_vars))
print("Time dtype:", ds.time.dtype)

# Select only FWI
fwi = ds["FWI"]

```

Many climate datasets:

- Store time as integers
- Or have broken CF metadata
- But **pygeoapi requires real datetime64 time axes**

- Try CF decoding
- “We guarantee a valid daily time axis — even if metadata is broken”
- This is defensive programming for operational services

Clipping to India boundary

```python
# ==================================================
# LOAD INDIA SHAPEFILE
# ==================================================
India = gpd.read_file(shp_path).to_crs("EPSG:4326")

# ==================================================
# SPATIAL METADATA (EPSG:4326)
# ==================================================
fwi = fwi.rio.write_crs("EPSG:4326", inplace=True)

fwi = fwi.rio.set_spatial_dims(
    x_dim="longitude",
    y_dim="latitude",
    inplace=True
)

# ==================================================
# CLIP TO INDIA (TIME PRESERVED)
# ==================================================
print("Clipping FWI cube to India...")
fwi_india = fwi.rio.clip(
    India.geometry,
    India.crs,
    drop=True
)

print("FWI India dims:", fwi_india.dims)

# ==================================================
# TIME SAFETY CHECK (CRITICAL FOR PYGEOAPI)
# ==================================================
assert np.issubdtype(
    fwi_india.time.dtype, np.datetime64
), "? Time axis is NOT datetime64!"
```

- Extracts only the Fire Weather Index variable.
- Drops unrelated variables early to save memory.

- Loads India boundary
- Reprojects it to WGS84 Raster and vector CRS must match before clipping.

- Many NetCDF files do not explicitly declare CRS Or declare it ambiguously
- rioxarray must know which dimensions are spatial

- Clips every time slice
- Preserves the time dimension
- Drops data outside India
- “This produces a true space-time cube over India.”

- If this fails, pygeoapi EDR will break Better to crash early than fail silently

Chunking data – Why it matters?

```python

# ==================================================
# CHUNKING (OPTIMAL FOR COVERAGE ACCESS)
# ==================================================
spatial_chunk = 100

fwi_india = fwi_india.chunk({
    "time": 1,
    "latitude": spatial_chunk,
    "longitude": spatial_chunk,
})

# ==================================================
# DATAARRAY ? DATASET
# ==================================================
ds_base = fwi_india.to_dataset(name="FWI")

# Drop stray CRS variable if present (avoids pygeoapi noise)
ds_base = ds_base.drop_vars("crs", errors="ignore")

```

- EDR queries often request:
  - One time
  - Small spatial region
- “We chunk for how users query, not how data was produced”
- pygeoapi expects datasets, not raw arrays
- Explicit variable naming helps metadata clarity
- Some libraries inject CRS as a variable and pygeoapi may misinterpret it
- So, dropping the stray variable

Writing into Zarr format & validation

```python

# ==================================================
# ZARR v3 WRITE (RAW CUBE)
# ==================================================
compressor = zarr.codecs.ZstdCodec(level=1)

encoding = {
    "FWI": {
        "dtype": "float32",
        "chunks": (1, spatial_chunk, spatial_chunk),
        "compressors": [compressor],
    }
}

print("Writing Zarr v3 (EPSG:4326, raw cube)...")

ds_base.to_zarr(
    zarr_out,
    mode="w",
    zarr_format=3,
    consolidated=False,   # correct for Zarr v3
    encoding=encoding,
)

# ==================================================
# SANITY CHECK
# ==================================================
print("Opening written Zarr...")
ds_check = xr.open_zarr(zarr_out, consolidated=False)
print(ds_check)

print("Time dtype:", ds_check.time.dtype)
print("Time values:", ds_check.time.values[:3])

```

Why Zstd Codec?

- Fast
- Good compression
- Widely supported

“Zarr v3 is ideal for cloud and HTTP-range access”

Check Confirms:

- Zarr structure is valid
- Time axis survived
- CRS survived

Important note

- For Linux environments, Zarr V3 is still in experimental stage
- Hence, stable Zarr v2 is used for demonstration

Visualization as confirmation

```
# ==================================================
# QUICK VISUAL TEST
# ==================================================
fwi_day10 = ds_check["FWI"].isel(time=9)

plt.figure(figsize=(7, 5))
fwi_day10.plot(cmap="Oranges", robust=True)
plt.title("FWI India - Day 10 (Zarr v3, raw cube)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()

print("? FWI Zarr v3 raw cube is READY for pygeoapi.")
```

Confirms:

- Orientation
- No flips
- Reasonable values
- “Always visually sanity-check geospatial data”

What this means

✓ Spatially correct
✓ Temporally valid
✓ Chunked for queries
✓ Cloud-optimized
✓ EDR-ready

Executing server script #1

- Open a new terminal in INSAT_GEO environment
- *cd* to test folder
- *python NC_ZARR_Win.py* (for Windows)
- *python NC_ZARR_Lin.py* (for Linux)

Local host experiment

Both client and server running in same system (Windows)

Setting up of server and client

- Open 2 terminals freshly – one for server and one for client

- Terminal #1
- STEP 1: NC_ZARR_Win.py - this takes netcdf file as input, does all required corrections, converts to Zarr v3 format and pushes to server using pygeoapi
  - For Linux, its Zarr version 2 (NC_ZARR_Lin.py)
  - Need to change names of python files

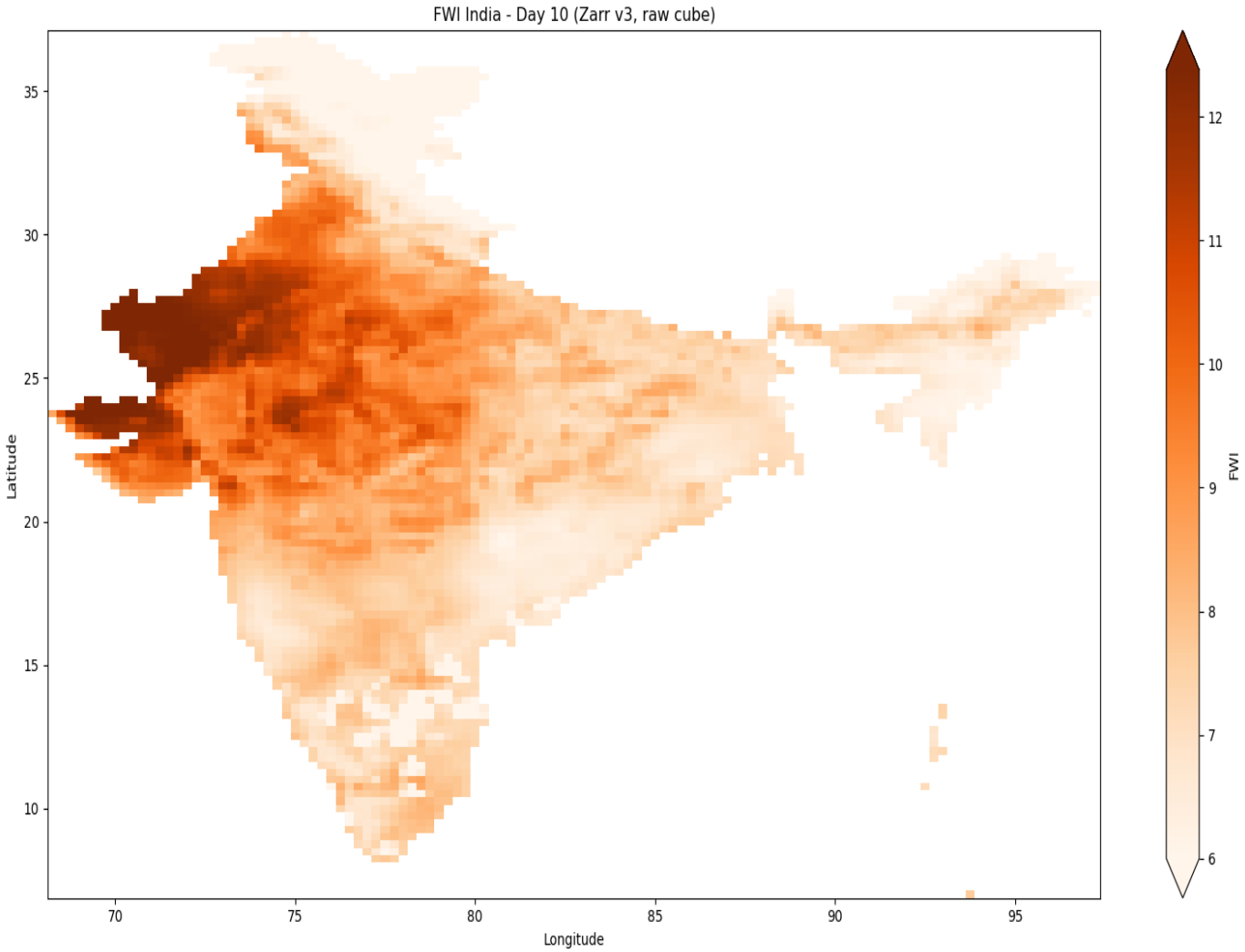
- STEP 2: Set configurations

- Terminal #2
- STEP 3: Win_Client_Query.py for Windows (or, Lin_Client_Query.py for Linux) is the client-side code to access the server(local host for now)

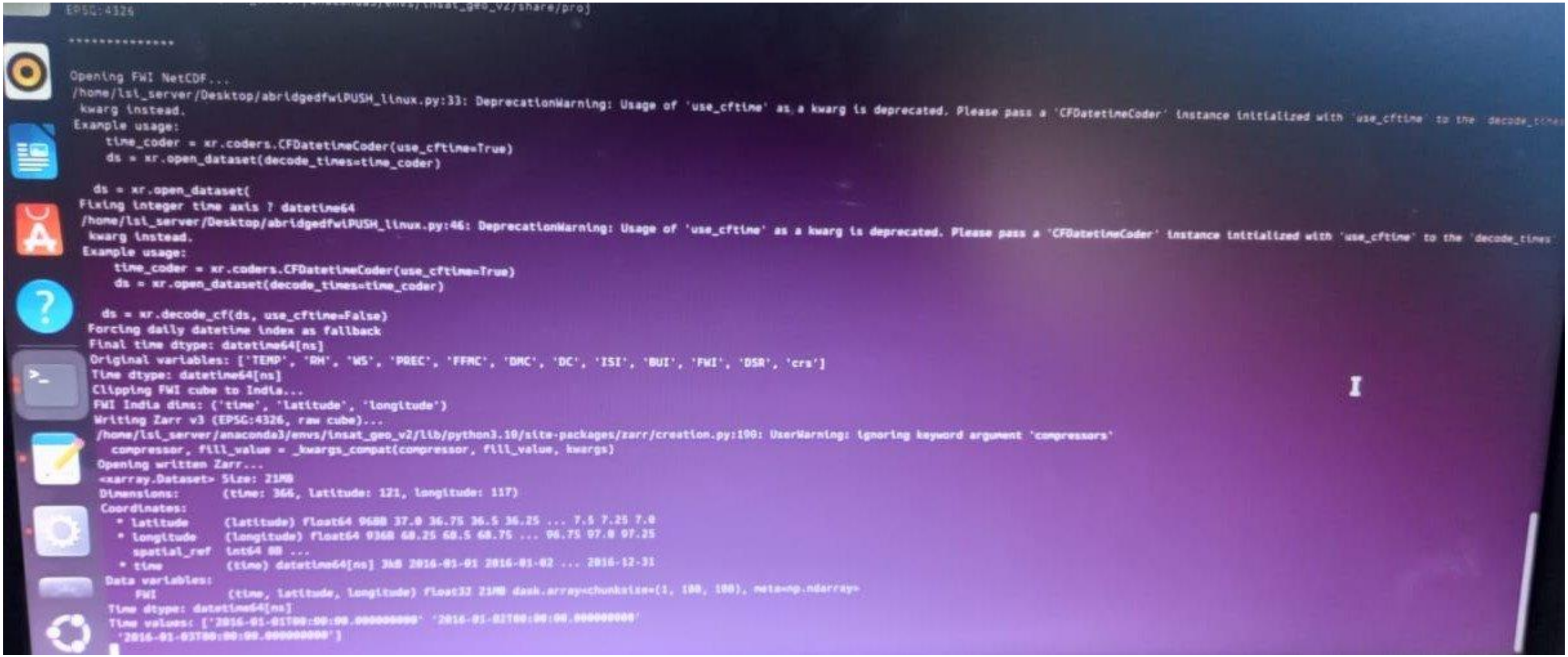
C:\Windows\system32\cmd.e

```
(INSAT_GEO) C:\Users\HP\Test>python abridgedfwiPUSH.py
PROJ version: 9.7.1
PROJ data dir: C:\Users\HP\.conda\envs\INSAT_GEO\Library\share\proj
EPSG:4326

**************

Opening FWI NetCDF...
C:\Users\HP\Test\abridgedfwiPUSH.py:42: DeprecationWarning: Usage of 'use_cftime' as a kwarg is deprecated. Please pass a 'CFDatetimeCoder' instance initialized with 'use_cftime' to the 'decode_times'
stead.
Example usage:
    time_coder = xr.coders.CFDatetimeCoder(use_cftime=True)
    ds = xr.open_dataset(decode_times=time_coder)

  ds = xr.open_dataset(
C:\Users\HP\Test\abridgedfwiPUSH.py:42: DeprecationWarning: Usage of 'use_cftime' as a kwarg is deprecated. Please pass a 'CFDatetimeCoder' instance initialized with 'use_cftime' to the 'decode_times'
stead.
Example usage:
    time_coder = xr.coders.CFDatetimeCoder(use_cftime=True)
    ds = xr.open_dataset(decode_times=time_coder)

  ds = xr.open_dataset(
Final time dtype: datetime64[ns]
Original variables: ['FWI']
Time dtype: datetime64[ns]
ERROR 1: PROJ: proj_create_from_database: Cannot find proj.db
C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pyogrio\core.py:35: RuntimeWarning: Could not detect PROJ data files. Set PROJ_LIB environment variable to the correct path.
  _init_proj_data()
Clipping FWI cube to India...
FWI India dims: ('time', 'latitude', 'longitude')
Writing Zarr v3 (EPSG:4326, raw cube)...
Opening written Zarr...
<xarray.Dataset> Size: 103MB
Dimensions:      (time: 1826, latitude: 121, longitude: 117)
Coordinates:
  * time         (time) datetime64[ns] 15kB 2014-01-01 2014-01-02 ... 2018-12-31
  * latitude     (latitude) float64 968B 37.0 36.75 36.5 36.25 ... 7.5 7.25 7.0
  * longitude    (longitude) float64 936B 68.25 68.5 68.75 ... 96.75 97.0 97.25
    spatial_ref  int64 8B ...
Data variables:
    FWI          (time, latitude, longitude) float32 103MB dask.array<chunksize=(1, 100, 100), meta=np.ndarray>
Time dtype: datetime64[ns]
Time values: ['2014-01-01T00:00:00.000000000' '2014-01-02T00:00:00.000000000'
 '2014-01-03T00:00:00.000000000']
```

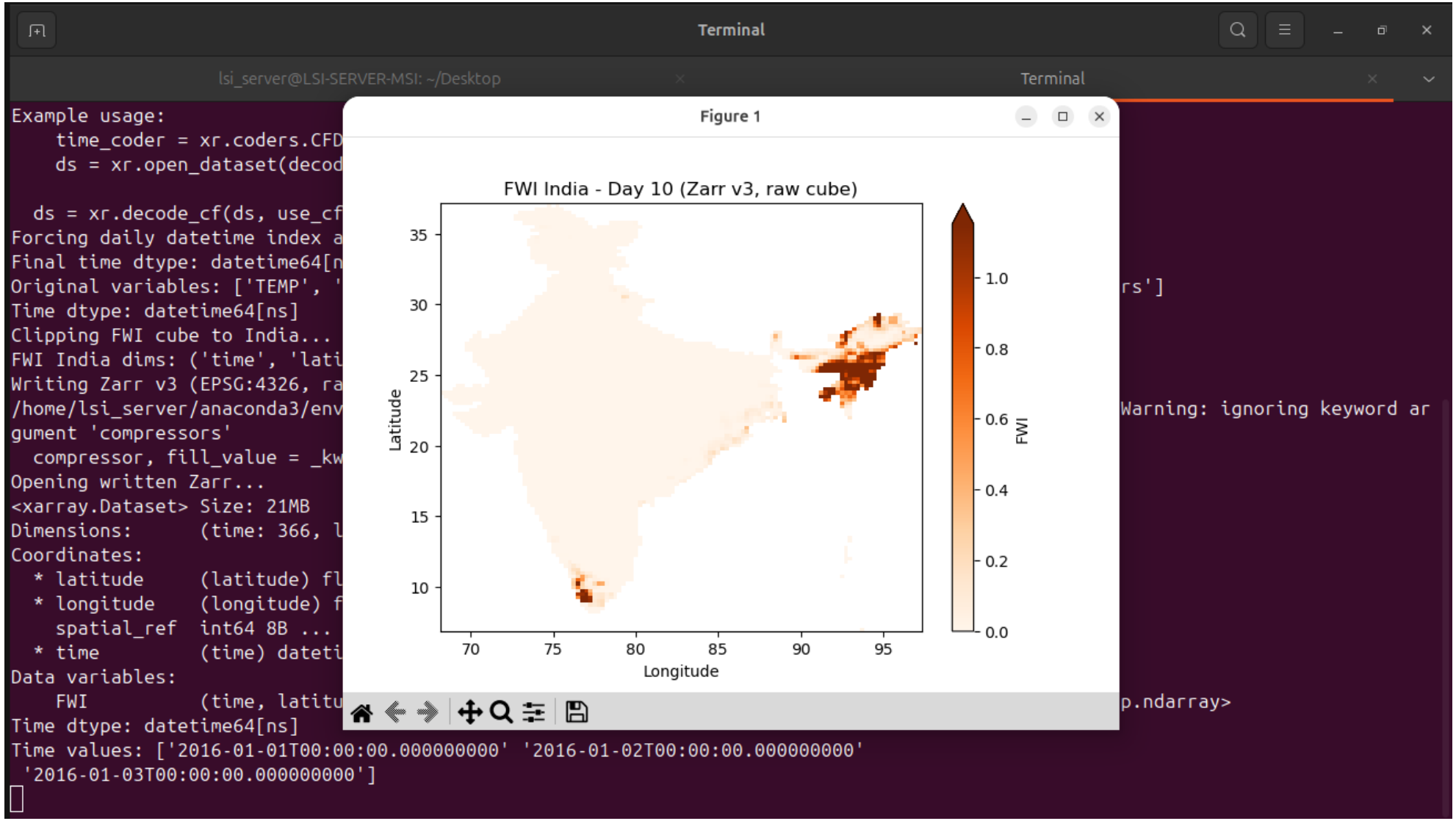
Visualization – India map showing Fire Weather Index values



Snapshot from Linux server

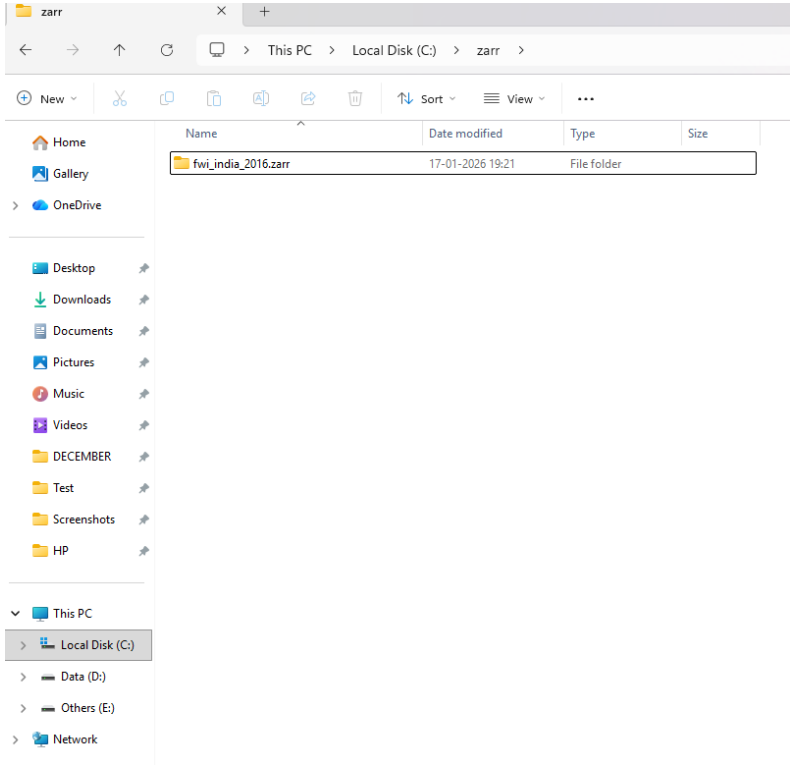


Map Output


Terminal
lsi_server@LSI-SERVER-MSI: ~/Desktop
Terminal
Example usage:
time_coder = xr.coders.CFD
ds = xr.open_dataset(decod
ds = xr.decode_cf(ds, use_cf
Forcing daily datetime index a
Final time dtype: datetime64[n
Original variables: ['TEMP', '
Time dtype: datetime64[ns]
Clipping FWI cube to India...
FWI India dims: ('time', 'lati
Writing Zarr v3 (EPSG:4326, ra
/home/lsi_server/anaconda3/env
gument 'compressors'
compressor, fill_value = _kw
Opening written Zarr...
<xarray.Dataset> Size: 21MB
Dimensions: (time: 366, l
Coordinates:
* latitude (latitude) fl
* longitude (longitude) f
spatial_ref int64 8B ...
* time (time) dateti
Data variables:
FWI (time, latitu
Time dtype: datetime64[ns]
Time values: ['2016-01-01T00:00:00.000000000' '2016-01-02T00:00:00.000000000'
'2016-01-03T00:00:00.000000000']
rs']
Warning: ignoring keyword ar
p.ndarray>
Figure 1
FWI India - Day 10 (Zarr v3, raw cube)
Latitude
Longitude
FWI
35
30
25
20
15
10
70
75
80
85
90
95
1.0
0.8
0.6
0.4
0.2
0.0

Validation

- Now, Zarr file is successfully created
- Check in C://zarr folder (or) /home/lsi_server/zarr



Server-Configuration Tasks

Configuring server – walk-through of pygeoapi-config.yml

- YAML and YML are the same thing — only the filename extension differs
- `.yml` exists only because early systems preferred **3-letter extensions**
- pygeoapi, Python, Docker, Kubernetes — treat them all identically

- **This YAML file tells pygeoapi:**
  - where to run the server
  - how to describe itself (metadata)
    - what datasets exist (resources)
  - where the data lives (Zarr)
  - how users can access it (Coverage + EDR)
- Think of it as **the contract between your data and the web**

"server" section

pygeoapi-config 3.yml

```yaml
# -*- coding: utf-8 -*-

server:
  bind:
    host: 0.0.0.0
    port: 5000
  url: http://localhost:5000

  # -------------------------
  # ENABLE OPENAPI GENERATION
  # -------------------------
  openapi: true
  openapi_url: /openapi

  language: en-US
  languages:
    - en-US

  map:                        # required for HTML UI
    url: https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png
    attribution: OpenStreetMap contributors

map:                          # also required at top-level
  url: https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png
  attribution: OpenStreetMap contributors

logging:
  level: DEBUG
```

- 0.0.0.0 → listen on **all network interfaces**
- 5000 → server port

- This is the **public-facing base URL u**sed in: OpenAPI, Landing page, Links returned to clients

- Enabling Auto-generated API documentation, Machine-readable API description

- OGC APIs support **internationalization -** Multiple languages can be added later

- Enables the **built-in pygeoapi web map using** OpenStreetMap tiles
- “This is only for visualization — not data delivery.”

- pygeoapi **requires** `map` at: `server.map`, top-level `map`

- Verbosity of logs

Metadata section for OGC compliance mandate

```yaml
metadata:
  identification:
    title: Local pygeoapi FWI Zarr Server
    description: pygeoapi serving Fire Weather Index (FWI) for India from local Zarr v3 datacube
    keywords:
      - pygeoapi
      - zarr
      - xarray
      - coverage
      - edr
      - fwi
      - fire_weather_index
      - india
      - datacube
    keywords_type: theme
    terms_of_service: none
    url: http://localhost:5000

  license:
    name: Public Domain
    url: https://creativecommons.org/publicdomain/zero/1.0/

  provider:
    name: Local
    url: http://localhost

  contact:
    name: Local User
    city: Localhost
    country: Local
```

- Human-readable description
- Appears on landing page
- Appears in /collections

- Keywords needed for Discovery, Catalogs, Search engines

- APIs must declare how data can be used

Resources are the core!

```yaml
resources:
  fwi_india:
    type: collection
    title: Fire Weather Index (FWI) - India (Zarr v3)
    description: India-clipped Fire Weather Index datacube served from local Zarr v3
    keywords:
      - fwi
      - fire_weather_index
      - india
      - zarr
      - datacube
      - wildfire

    extents:
      spatial:
        bbox:
          - [68.0, 6.0, 98.0, 37.0]
        crs: http://www.opengis.net/def/crs/OGC/1.3/CRS84
      temporal:
        begin: 2016-01-01T00:00:00Z
        end: 2016-12-31T00:00:00Z
```

- Each entry = **one dataset**
- fwi_india becomes: /collections/fwi_india

- A collection is a logical dataset, not a file

- Deals with: Geographic coverage, Enables map previews, Enables bounding-box validation

- Clients can discover what time range exists without querying data

Coverage and EDR providers

```yaml
providers:
  # -------------------------
  # OGC API - Coverages
  # -------------------------
  - type: coverage
    name: xarray
    data: C:/zarr/fwi_india_2016.zarr
    variable: FWI
    x_field: longitude
    y_field: latitude
    time_field: time
    options:
      engine: zarr
      consolidated: false

  # -------------------------
  # OGC API - EDR
  # -------------------------
  - type: edr
    name: xarray-edr
    data: C:/zarr/fwi_india_2016.zarr
    variable: FWI
    x_field: longitude
    y_field: latitude
    time_field: time
    options:
      engine: zarr
      consolidated: false
```

This is where your **Zarr cube becomes an API**

Coverages section enables:

- /collections/fwi_india/coverage
- Grid-style access
- Raster semantics

- We point to a Zarr store, not a NetCDF file

- Tells pygeoapi how dimensions map to space and time
- Must match your preprocessing script exactly

EDR section enables:

- `/collections/fwi_india/cube`
- `/edr/position`
- `/edr/area`

EDR is for *querying*, not downloading

- Matches Zarr v3 writing style
- Prevents metadata mismatch

Executing server script #2

- Open a new Terminal in INSAT_GEO environment
- *cd* to Test folder
- Set/Export Configurations and start pygeoapi server

Code walk-through – Client on Windows

Acts as a client to a pygeoapi EDR service
– queries Fire Weather Index (FWI) data
– reconstructs spatial grids from EDR responses
– visualizes a single day as a map
– computes and plots a multi-day time series

“This script shows how users actually *consume* an EDR data cube.”

Library imports

```python
import requests
import numpy as np
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
```

Library	Role
requests	Makes HTTP calls to pygeoapi
numpy	Builds numeric axes
pandas	Tabular reconstruction
geopandas	Spatial plotting
matplotlib	Visualization

From where (spatial) and *when (temporal)* do we want data subset?

```
# ==================================================
# CONFIG
# ==================================================
URL_ROOT = "http://localhost:5000"
COLLECTION = "fwi_india"

BBOX = "78.25,17.25,78.75,17.75"
REQUESTED_TIME = "2016-03-15T00:00:00Z"
FWI_TIMERANGE = "2016-03-01/2016-03-31"

```

- URL_ROOT: where pygeoapi is running
- COLLECTION: the dataset being queried

- Bounding Box carries location extents
- Single and Multiday query are demonstrated

Packing request for single day Environmental Data Retrieval (EDR) cube

```python
# ==================================================
# 1. REQUEST SINGLE-DAY EDR CUBE
# ==================================================
cube_url = f"{URL_ROOT}/collections/{COLLECTION}/cube"

params = {
    "bbox": BBOX,
    "datetime": REQUESTED_TIME,
    "parameter-name": "FWI",
    "f": "json"
}

response = requests.get(cube_url, params=params)
response.raise_for_status()
cube = response.json()
```

- This is an **EDR Cube query**
- It returns gridded data over space and time
- "EDR queries are *parameterized URLs*, not file downloads."

- Response cube contains:
  - Metadata (axes)
  - Values (FWI)
  - Domain description

Detecting axes

```python

# ==================================================
# 2. DETECT AXES (ROBUST)
# ==================================================
axes = cube["domain"]["axes"]

x_name = "longitude" if "longitude" in axes else "x"
y_name = "latitude" if "latitude" in axes else "y"

x_axis = axes[x_name]
y_axis = axes[y_name]

# time may be collapsed
if "time" in axes:
    t_axis = axes["time"]
    time_vals = pd.date_range(
        start=t_axis["start"],
        periods=t_axis["num"],
        freq="D"
    )
else:
    time_vals = [pd.to_datetime(REQUESTED_TIME)]

# ==================================================
# 3. BUILD AXIS VALUES
# ==================================================
x_vals = np.linspace(x_axis["start"], x_axis["stop"], x_axis["num"])
y_vals = np.linspace(y_axis["start"], y_axis["stop"], y_axis["num"])

```

- EDR **does not guarantee axis names**
- So, a good client must adapt to different EDR implementations

Building axes and extracting values

```
# ==================================================
# 3. BUILD AXIS VALUES
# ==================================================
x_vals = np.linspace(x_axis["start"], x_axis["stop"], x_axis["num"])
y_vals = np.linspace(y_axis["start"], y_axis["stop"], y_axis["num"])

# ==================================================
# 4. EXTRACT VALUES
# ==================================================
fwi_vals = cube["ranges"]["FWI"]["values"]
```

What this does

- Converts axis metadata into **actual coordinate arrays**

- EDR gives:
  - start
  - stop
  - number of points

- We reconstruct the grid numerically
  - "EDR returns *compact metadata*, not full coordinate arrays."

- Important concept
  - Values arrive as a **flat 1D list**
  - Ordering is implicit, not explicit
  - "EDR prioritizes compact transmission over convenience

Grid reconstruction

```python
# ==================================================
# 5. RECONSTRUCT GRID (SAFE)
# ==================================================
records = []
idx = 0
n_vals = len(fwi_vals)

for t in time_vals:
    for y in y_vals:
        for x in x_vals:
            if idx >= n_vals:
                break
            records.append({
                "time": t,
                "longitude": x,
                "latitude": y,
                "FWI": fwi_vals[idx]
            })
            idx += 1

df = pd.DataFrame(records)

gdf = gpd.GeoDataFrame(
    df,
    geometry=gpd.points_from_xy(df.longitude, df.latitude),
    crs="EPSG:4326"
)

print("GeoDataFrame preview (single day):")
print(gdf.head())
```

- We rebuild the **space–time cube explicitly**
- We walk through: time → latitude → longitude
- Each value is mapped to a real location

- EDR clients must know how to reassemble grids
- This mirrors how NetCDF or Zarr cubes are structured internally

- Creating GIS Objects (GeoDataFrame)
  - Adds spatial meaning
  - Enables GIS-style plotting
  - Allows future spatial operations
  - We move from *web response* → *GIS object*

Single day visualization

```
# ==================================================
# 6. PLOT SINGLE-DAY MAP
# ==================================================
fig, ax = plt.subplots(figsize=(7, 6))
gdf.plot(column="FWI", cmap="Oranges", markersize=80, legend=True, ax=ax)
ax.set_title(f"Fire Weather Index (FWI) - {pd.to_datetime(REQUESTED_TIME).date()}")
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")
plt.tight_layout()
plt.show()
```

What this shows

- Spatial distribution of FWI
- For **one day**
- Over the requested bbox

Multi-day EDR cube

```python

# ==================================================
# 7. MULTI-DAY EDR CUBE (PRISM-LIKE FEATURE)
# ==================================================
params_range = {
    "bbox": BBOX,
    "datetime": FWI_TIMERANGE,
    "parameter-name": "FWI",
    "f": "json"
}

response_range = requests.get(cube_url, params=params_range)
response_range.raise_for_status()
cube_range = response_range.json()

axes_r = cube_range["domain"]["axes"]

x_name = "longitude" if "longitude" in axes_r else "x"
y_name = "latitude" if "latitude" in axes_r else "y"

x_axis = axes_r[x_name]
y_axis = axes_r[y_name]

x_vals = np.linspace(x_axis["start"], x_axis["stop"], x_axis["num"])
y_vals = np.linspace(y_axis["start"], y_axis["stop"], y_axis["num"])

# time axis may or may not exist
if "time" in axes_r:
    t_axis = axes_r["time"]
    time_vals = pd.date_range(
        start=t_axis["start"],
        periods=t_axis["num"],
        freq="D"
    )
else:
    start, end = FWI_TIMERANGE.split("/")
    time_vals = pd.date_range(start=start, end=end, freq="D")

fwi_vals = cube_range["ranges"]["FWI"]["values"]

```

What's new here

- We request a **time range**
- Same spatial bbox
- Same parameter

This turns the cube into: **Space × Time data**

EDR servers may:

- Explicitly return time axis
- Or implicitly encode time in values
- Clients must never assume the server behaves perfectly

Reconstruct range cube

```python

# ==================================================
# 8. RECONSTRUCT RANGE CUBE (SAFE)
# ==================================================
records = []
idx = 0
n_vals = len(fwi_vals)
nx = len(x_vals)
ny = len(y_vals)
n_space = nx * ny
n_time = max(1, n_vals // n_space)

time_vals = time_vals[:n_time]

for t in time_vals:
    for y in y_vals:
        for x in x_vals:
            if idx >= n_vals:
                break
            records.append({
                "time": t,
                "longitude": x,
                "latitude": y,
                "FWI": fwi_vals[idx]
            })
            idx += 1

df_cube = pd.DataFrame(records)

```

What this calculates

- How many time steps exist
- Based on total values received

This avoids:

- Index overruns
- Misaligned cubes

“We infer cube shape safely from metadata + values”

EDR enabled analysis

```python
# ==================================================
# 9. DAILY MEAN FWI TIME SERIES
# ==================================================
daily_mean_fwi = (
    df_cube
    .groupby("time")["FWI"]
    .mean()
    .reset_index()
)

print("\nDaily mean FWI over bbox:")
print(daily_mean_fwi.head())

# ==================================================
# 10. PLOT TIME SERIES
# ==================================================
plt.figure(figsize=(8, 4))
plt.plot(daily_mean_fwi["time"], daily_mean_fwi["FWI"], marker="o")
plt.title("Daily Mean Fire Weather Index (FWI): Hyderabad neighbourhood")
plt.xlabel("Date")
plt.ylabel("FWI")
plt.grid(True)
plt.tight_layout()
plt.show()
```

- Aggregates spatial data into **one number per day**
  - Risk assessment
  - Trend analysis
  - Dashboards
- What this shows
  - Temporal evolution of fire risk
  - Over a specific region

Summary

- This script demonstrates how a user queries an EDR service, reconstructs a spatial–temporal data cube, and converts it into maps and time series for analysis

Prepare to run the server…

- *pip install pygeoapi*

- Windows:
  - set PYGEOAPI_CONFIG=C:\Users\Lenovo\Test\pygeoapi-config.yml
  - set PYGEOAPI_OPENAPI=C:\Users\Lenovo\Test\openapi.yml
  - pygeoapi serve

- Linux:
  - export PYGEOAPI_CONFIG=/home/lsi_server/Desktop/pygeoapi-config.yml
  - export PYGEOAPI_OPENAPI=/home/lsi_server/Desktop/openapi.yml

```
(INSAT_GEO) C:\Users\HP\Test>set PYGEOAPI_CONFIG=C:\Users\HP\Test\pygeoapi-config.yml

(INSAT_GEO) C:\Users\HP\Test>set PYGEOAPI_OPENAPI=C:\Users\HP\Test\openapi.yml

(INSAT_GEO) C:\Users\HP\Test>pygeoapi serve
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\log.py:124} DEBUG - Logging initialized
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\process\manager\base.py:511} INFO - Starting dummy manager
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:111} DEBUG - Plugins: {'Dummy': 'pygeoapi.process.manager.dummy.DummyManager', 'MongoDB': 'pygeoapi
r.mongodb_.MongoDBManager', 'TinyDB': 'pygeoapi.process.manager.tinydb_.TinyDBManager', 'PostgreSQL': 'pygeoapi.process.manager.postgresql.PostgreSQLManager'}
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:123} DEBUG - package name: pygeoapi.process.manager.dummy
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:124} DEBUG - class name: DummyManager
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\api\__init__.py:572} INFO - Process manager plugin loaded
 * Serving Flask app 'pygeoapi.flask_app'
 * Debug mode: on
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\werkzeug\_internal.py:97} INFO - WARNING: This is a development server. Do not use it in a production deployment. Use
GI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.2.129.45:5000
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\werkzeug\_internal.py:97} INFO - Press CTRL+C to quit
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\werkzeug\_internal.py:97} INFO -  * Restarting with stat
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\log.py:124} DEBUG - Logging initialized
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\process\manager\base.py:511} INFO - Starting dummy manager
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:111} DEBUG - Plugins: {'Dummy': 'pygeoapi.process.manager.dummy.DummyManager', 'MongoDB': 'pygeoapi
r.mongodb_.MongoDBManager', 'TinyDB': 'pygeoapi.process.manager.tinydb_.TinyDBManager', 'PostgreSQL': 'pygeoapi.process.manager.postgresql.PostgreSQLManager'}
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:123} DEBUG - package name: pygeoapi.process.manager.dummy
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\plugin.py:124} DEBUG - class name: DummyManager
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\pygeoapi\api\__init__.py:572} INFO - Process manager plugin loaded
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\werkzeug\_internal.py:97} WARNING -  * Debugger is active!
[2026-01-17T19:39:18Z] {C:\Users\HP\.conda\envs\INSAT_GEO\Lib\site-packages\werkzeug\_internal.py:97} INFO -  * Debugger PIN: 136-011-961
```

Terminal

lsi_server@LSI-SERVER-MSI: ~/Desktop | Terminal | lsi_server@LSI-SERVER-MSI: ~/Desktop/mewan

```
(insat_geo_v2) lsi_server@LSI-SERVER-MSI:~/Desktop$ pygeoapi serve
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/log.py:124} DEBUG - Logging initialized
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/process/manager/base.py:511} INFO - Starting dummy manager
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:111} DEBUG - Plugins: {'Dummy': 'pygeoapi.process.manager.dummy.Du
mmyManager', 'MongoDB': 'pygeoapi.process.manager.mongodb_.MongoDBManager', 'TinyDB': 'pygeoapi.process.manager.tinydb_.TinyDBManager', 'PostgreSQL': 'pygeoapi.process.manager.postgre
sql.PostgreSQLManager'}
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:123} DEBUG - package name: pygeoapi.process.manager.dummy
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:124} DEBUG - class name: DummyManager
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/api/__init__.py:572} INFO - Process manager plugin loaded
 * Serving Flask app 'pygeoapi.flask_app'
 * Debug mode: on
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/werkzeug/_internal.py:97} INFO - WARNING: This is a development server. Do not use it
 in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.2.57.44:5000
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/werkzeug/_internal.py:97} INFO - Press CTRL+C to quit
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/werkzeug/_internal.py:97} INFO -  * Restarting with stat
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/log.py:124} DEBUG - Logging initialized
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/process/manager/base.py:511} INFO - Starting dummy manager
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:111} DEBUG - Plugins: {'Dummy': 'pygeoapi.process.manager.dummy.Du
mmyManager', 'MongoDB': 'pygeoapi.process.manager.mongodb_.MongoDBManager', 'TinyDB': 'pygeoapi.process.manager.tinydb_.TinyDBManager', 'PostgreSQL': 'pygeoapi.process.manager.postgre
sql.PostgreSQLManager'}
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:123} DEBUG - package name: pygeoapi.process.manager.dummy
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/plugin.py:124} DEBUG - class name: DummyManager
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/pygeoapi/api/__init__.py:572} INFO - Process manager plugin loaded
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/werkzeug/_internal.py:97} WARNING -  * Debugger is active!
[2026-01-20T15:54:34Z] {/home/lsi_server/anaconda3/envs/insat_geo_v2/lib/python3.10/site-packages/werkzeug/_internal.py:97} INFO -  * Debugger PIN: 974-690-122
```

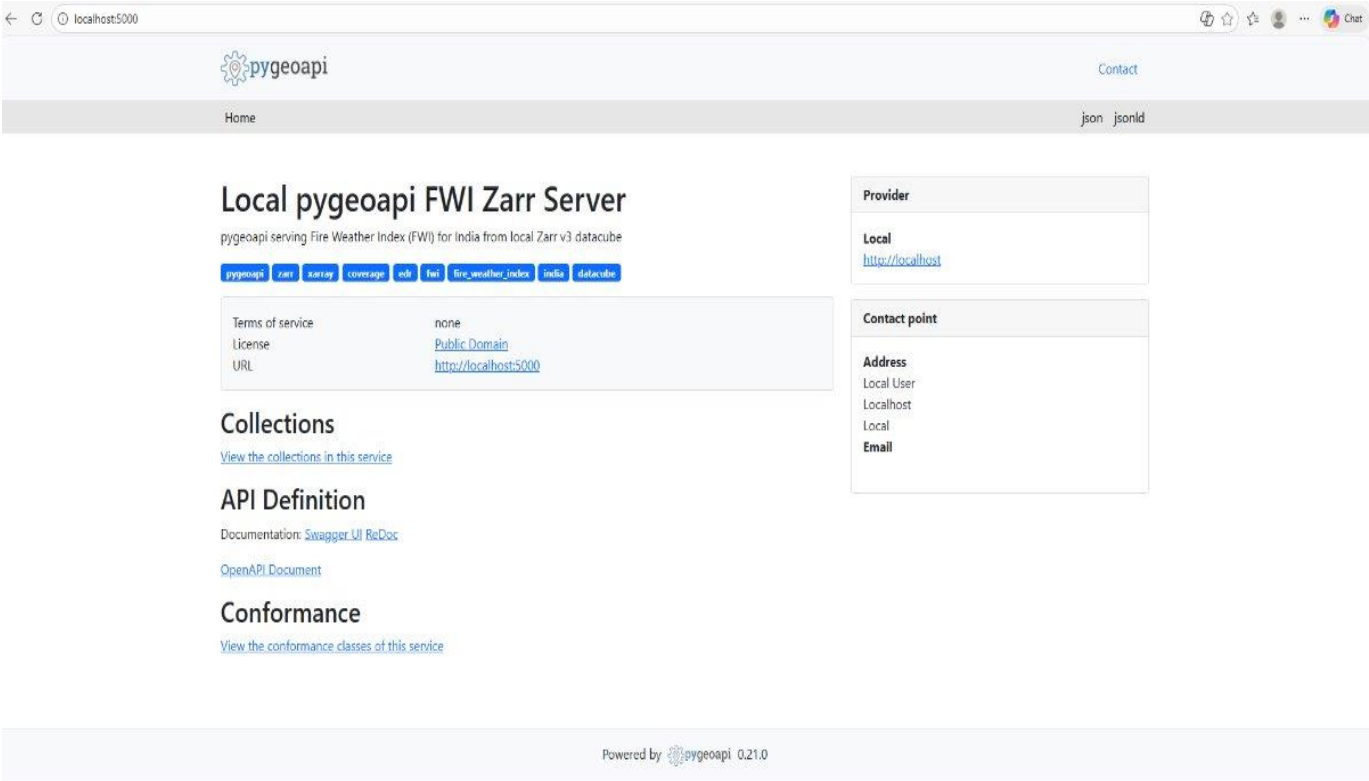
Client Tasks

Client Types

- Two types of clients
  - Browser
  - Python API based access

Option #1: Sample web query

- Open any usual browser (Firefox/Edge/Chrome)
- http://localhost:5000/



Try more links...

- http://localhost:5000/collections?f=html
- http://localhost:5000/collections/fwi_india
- http://localhost:5000/collections/fwi_india?f=json

- JSON: http://localhost:5000/collections/fwi_india/position?coords=POINT(79.4867%2021.3850)
- HTML: http://localhost:5000/collections/fwi_india/position?coords=POINT(78.4867%2017.3850)&f=html

- Cube Trial:
- http://localhost:5000/collections/fwi_india/cube?bbox=78.25,17.25,78.75,17.75&datetime=2016-03-...
- http://localhost:5000/collections/fwi_india/cube?bbox=78.25,17.25,78.75,17.75&datetime=2016-01-...

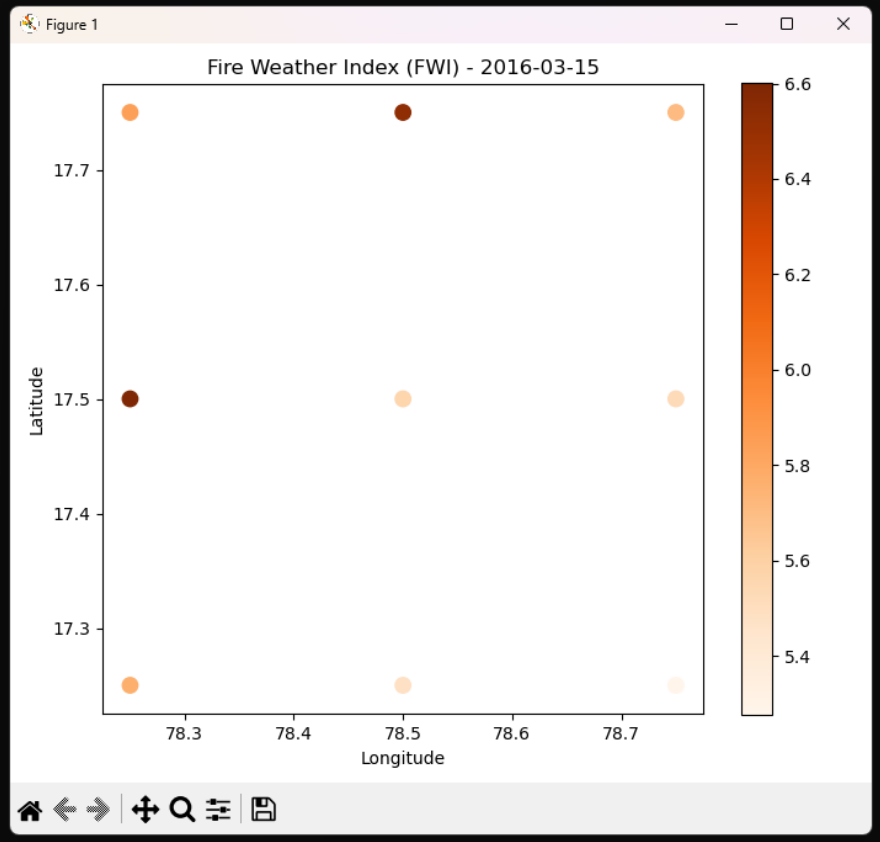
- Coverage - Json:
- http://localhost:5000/collections/fwi_india/coverage?subset=longitude(78.25,78.75)&subset=latit...

Option #2: Python code-based client

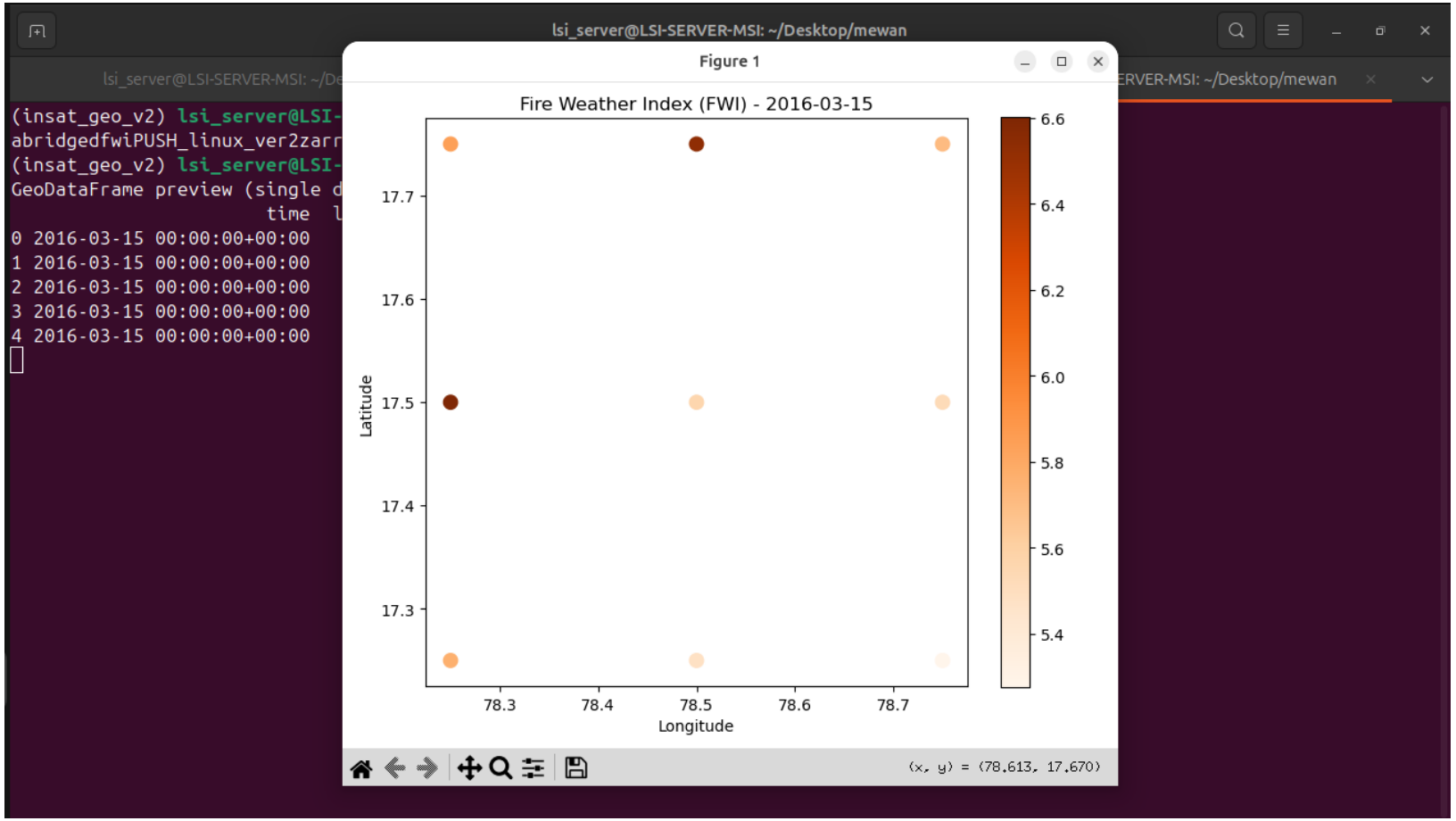
- Open new terminal exclusively for client
- *cd* to Test folder
- *python Win_Client_Query.py* (for Windows)
- *python Lin_Client_Query.py* (for Linux)

Output #1 – Single day query (Windows)

```
(INSAT_GEO) C:\Users\HP\Test>python fwi_FULLSERVICE.py
GeoDataFrame preview (single day):
                       time  longitude  latitude       FWI            geometry
0 2016-03-15 00:00:00+00:00      78.25     17.75  5.844927  POINT (78.25 17.75)
1 2016-03-15 00:00:00+00:00      78.50     17.75  6.534139   POINT (78.5 17.75)
2 2016-03-15 00:00:00+00:00      78.75     17.75  5.707436  POINT (78.75 17.75)
3 2016-03-15 00:00:00+00:00      78.25     17.50  6.601877   POINT (78.25 17.5)
4 2016-03-15 00:00:00+00:00      78.50     17.50  5.563274    POINT (78.5 17.5)
```

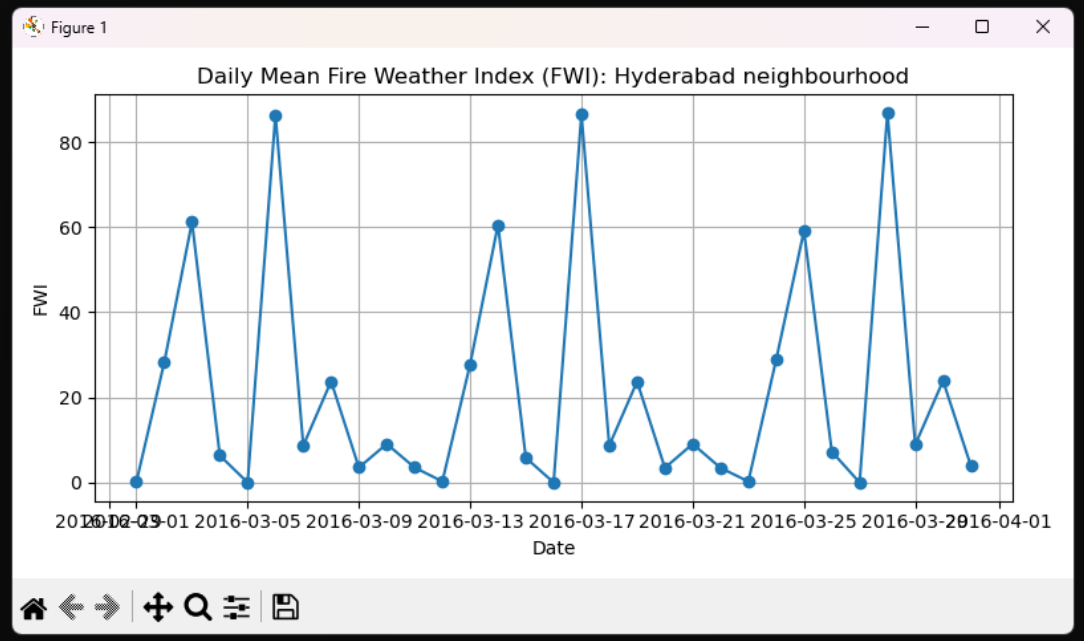


Output #1 – Single day query (Linux)

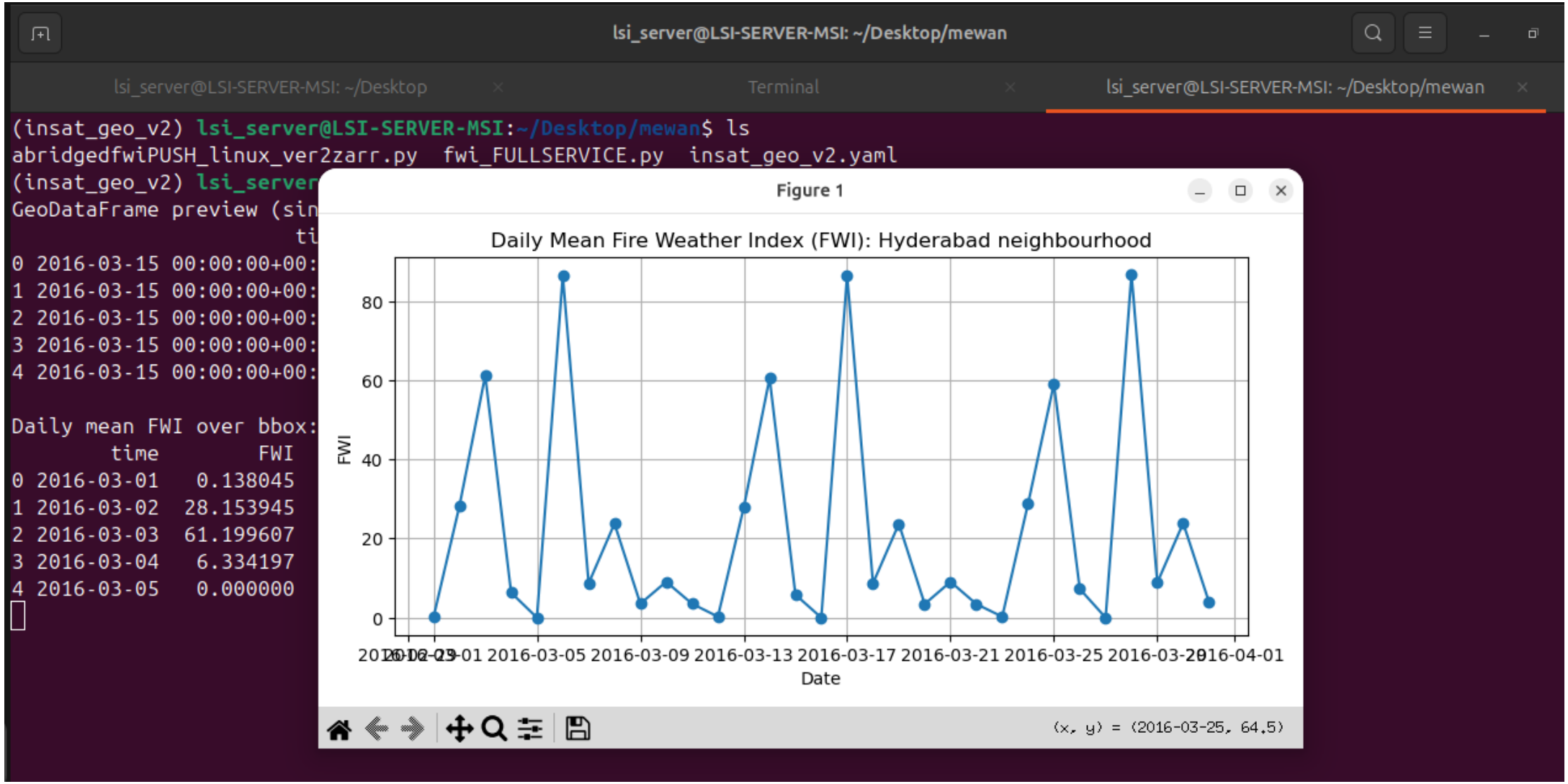


Output #2 – Multiday query (Windows)

```
(INSAT_GEO) C:\Users\HP\Test>python fwi_FULLSERVICE.py
GeoDataFrame preview (single day):
                       time  longitude  latitude       FWI             geometry
0 2016-03-15 00:00:00+00:00      78.25     17.75  5.844927  POINT (78.25 17.75)
1 2016-03-15 00:00:00+00:00      78.50     17.75  6.534139   POINT (78.5 17.75)
2 2016-03-15 00:00:00+00:00      78.75     17.75  5.707436  POINT (78.75 17.75)
3 2016-03-15 00:00:00+00:00      78.25     17.50  6.601877   POINT (78.25 17.5)
4 2016-03-15 00:00:00+00:00      78.50     17.50  5.563274    POINT (78.5 17.5)

Daily mean FWI over bbox:
        time        FWI
0 2016-03-01   0.138045
1 2016-03-02  28.153945
2 2016-03-03  61.199607
3 2016-03-04   6.334197
4 2016-03-05   0.000000
```



Output #2 – Multiday query (Linux)



Console dump

```
lsi_server@LSI-SERVER-MSI: ~/Desktop/mewan

lsi_server@LSI-SERVER-MSI: ~/Desktop    Terminal    lsi_server@LSI-SERVER-MSI: ~/Desktop/mewan

(insat_geo_v2) lsi_server@LSI-SERVER-MSI:~/Desktop/mewan$ ls
abridgedfwiPUSH_linux_ver2zarr.py  fwi_FULLSERVICE.py  insat_geo_v2.yaml
(insat_geo_v2) lsi_server@LSI-SERVER-MSI:~/Desktop/mewan$ python3 fwi_FULLSERVICE.py
GeoDataFrame preview (single day):
                      time  longitude  latitude       FWI             geometry
0 2016-03-15 00:00:00+00:00      78.25     17.75  5.844927  POINT (78.25 17.75)
1 2016-03-15 00:00:00+00:00      78.50     17.75  6.534139   POINT (78.5 17.75)
2 2016-03-15 00:00:00+00:00      78.75     17.75  5.707436  POINT (78.75 17.75)
3 2016-03-15 00:00:00+00:00      78.25     17.50  6.601877   POINT (78.25 17.5)
4 2016-03-15 00:00:00+00:00      78.50     17.50  5.563274    POINT (78.5 17.5)

Daily mean FWI over bbox:
        time        FWI
0 2016-03-01   0.138045
1 2016-03-02  28.153945
2 2016-03-03  61.199607
3 2016-03-04   6.334197
4 2016-03-05   0.000000
(insat_geo_v2) lsi_server@LSI-SERVER-MSI:~/Desktop/mewan$
```

Testing over dedicated Windows/Linux server

Run Command Prompt (cmd) in Windows in administrator mode

Type ipconfig

Setting up Windows server – Get IPv4 address

```
Command Prompt

Microsoft Windows [Version 10.0.26200.7623]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : iiit.ac.in

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : iiit.ac.in
   Link-local IPv6 Address . . . . . : fe80::b8e8:16d9:b650:a53c%11
   IPv4 Address. . . . . . . . . . . : 10.2.129.45
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . : 10.2.128.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\HP>
```

Editing client code with this IP address

- URL_ROOT – edit
- Replace http://localhost:5000 with

  http://10.129.45:5000

- Note: This will change based on the wifi
- Mobile hotspots issued IP address **WILL NOT WORK**
- Repeat the steps

```
# ==================================================
# CONFIG
# ==================================================
URL_ROOT = "http://localhost:5000"
COLLECTION = "fwi_india"

BBOX = "78.25,17.25,78.75,17.75"
REQUESTED_TIME = "2016-03-15T00:00:00Z"
FWI_TIMERANGE = "2016-03-01/2016-03-31"

```

Testing over dedicated Linux server

Setting up Linux server

- Open New Terminal
- Same steps as for Windows but use ifconfig command instead of ipconfig to get IPv4 address

Editing client code with this IP address

- URL_ROOT – same editing process as described before

- Repeat the steps

What if client and server are not in the same Wi-Fi network?

- Grok code - Explain

Grok code

- *pip install pyngrok* **in a new terminal**

- Type below code and save as grok.py and run it:

```
from pyngrok import ngrok
url = ngrok.connect(5000)
print(url)
```

- This will output a temporary cloud flare link

DO NOT CLOSE THIS TERMINAL or STOP THE PROCESS AT ANY COST.

Otherwise, the free trial URL generated in that session will expire

Editing the client code

- URL_ROOT: <grok URL> --- Same editing process in the code

- Sample Grok link using Cloudflare server:

  https://perhaps-score-mat-iiith.trycloudflare.com

- Repeat the steps mentioned earlier to execute

Experiments to try…

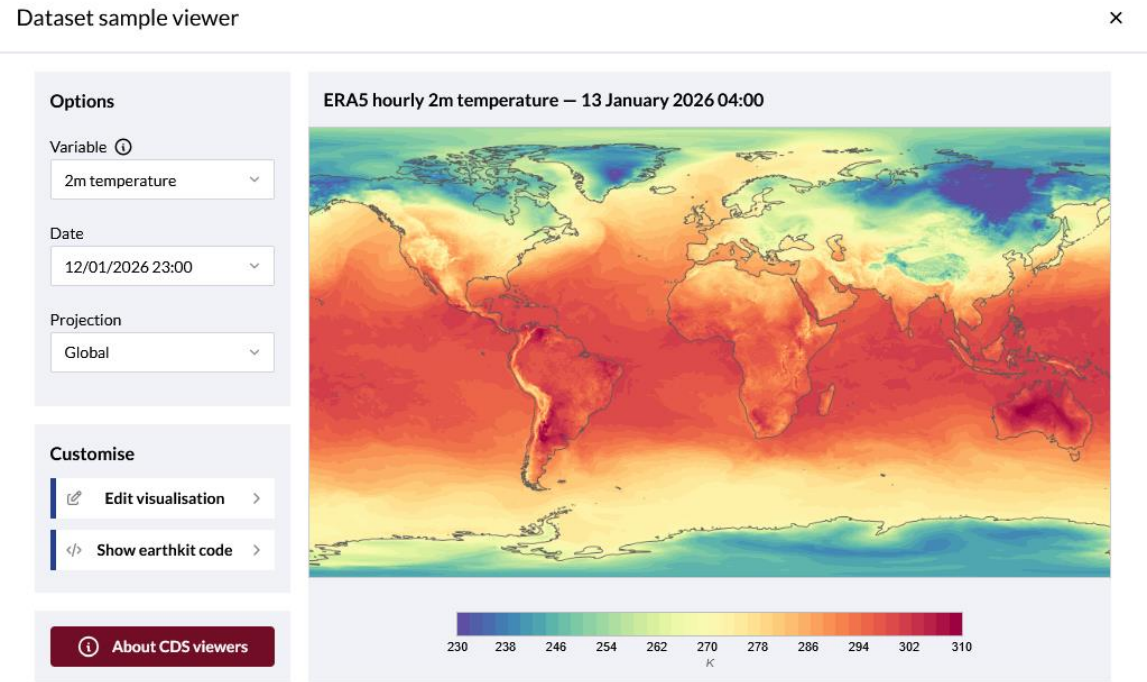
1. Download a variable of your choice, say temperature, using code from ECMWF ERA5 Climate Reanalysis Dataset and try from scratch
2. Quirky Visualizations – Fire bar chart

Case Study: Temperature

- **The Climate Data Store (CDS) Application Program Interface (API)** is a service providing programmatic access to CDS
  - https://cds.climate.copernicus.eu/how-to-api

- ERA5 hourly data on single levels from 1940 to present
  - https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=overview
- ARD:
- Please note that a dedicated catalogue entry for this dataset, post-processed and stored in Analysis Ready Cloud Optimized (ARCO) format (Zarr), is available for optimised time-series retrievals (i.e. for retrieving data from selected variables for a single point over an extended period of time in an efficient way). You can discover it here
  - https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels-timeseries?tab=overview

- https://jupyterlab.readthedocs.io/en/latest/

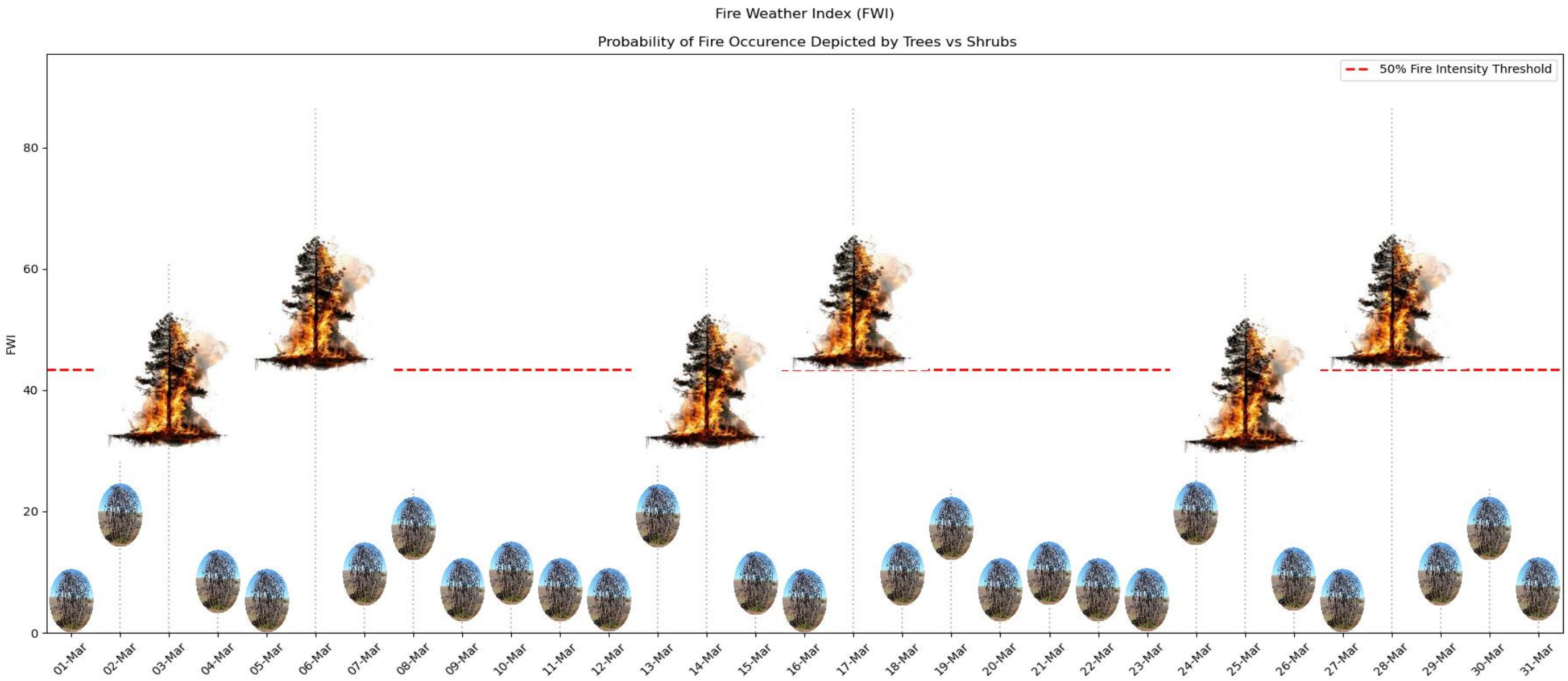


Cdstestbox code

cdstest.py

cdstestbox.py

Quirky visualization...!



What’s next…?

- Publish your customized processes through pygeo api
- For example, you want to access temperature data from USGS datasets and forestry data from ECMWF datasets and then unify it in your product, then, such mixed dataset hosting is also possible with the use of this Zarr + pygeoapi framework!
- An environment NGO in charge of doing animal census, can mix and match datasets, to see how heatwave can affect strays living on streets and therefore can cause disease outbreaks – then help visualize it to request government to make policies --- such things can be registered using your own yml config file.

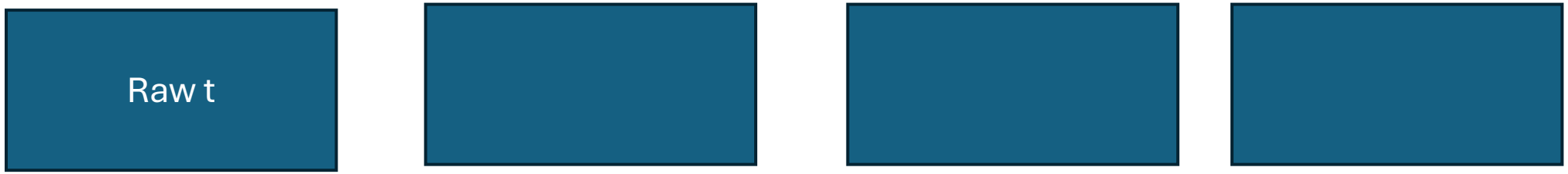
List of References

- https://medium.com/geobeyond/how-to-get-started-with-pygeoapi-79f77a894249
- https://docs.pygeoapi.io/en/stable/publishing/ogcapi-coverages.html
- https://docs.pygeoapi.io/en/0.20.0/data-publishing/ogcapi-edr.html
- https://water.usgs.gov/catalog/usecases/77aa3413-3993-4475-93cd-e4f7b5ceecc9/

Questions?

malini.krishnan@research.iiit.ac.in

Feedback!


Raw t