Final Report Builder – Python Practice & Capstone (Print-Ready)

This notebook is designed to help you create a **single, well-organised report** of all your work:

- Exercises 01–04
- Combined Exam Notebook (if used)
- Full Capstone Assignment

At the end, you will **export this notebook as a PDF** and print it / place it in your journal.

How to Use This Notebook

1. Work through each section in order.
2. Fill in the text sections in your own words.
3. Where indicated, **run code cells** to regenerate important plots.
4. Check that all plots and text are visible and clear.
5. When finished:
    - In Colab:
        - `File → Print` → choose **Save as PDF**
    - Or `File → Download` → `Download PDF` (if available)
6. Save the PDF with a filename like:
   `YourName_Python_Report.pdf`

> **Important:** This report should reflect **your own work**. Do not copy text or code from classmates.

---

1. Student & Course Information

Fill in your details below.

```
# This cell automatically prints today's date.
from datetime import datetime
from IPython.display import Markdown

today = datetime.today().strftime("%d %B %Y")
Markdown(f"**Date of Report Completion (auto):** {today}")
```

Date of Report Completion (auto): 22 December 2025

Name: Vedika Kavlekar

Student ID / Roll No.:202200737

Programme: Master of Science (Remote Sensing) and (GIS)

Course Title: Applicable for RSG-5026, RSG-5028, RSG-5222

Semester / Academic Year: 2nd Semister 2025-26

Instructor: Dr. ________________

(If needed) Date of Report Completion (manual override):

---

2. Overview of Work Completed

In this section, you will summarise which notebooks and assignments you have completed.

2.1 – List of Completed Notebooks

Fill in the table (edit in Markdown):

Notebook / Assignment	Status (Completed / Partial / Not done)	Comments
01 – Python Fundamentals	Completed	
02 – Lists, Loops, and Conditions	Completed	
03 – Functions and Plotting	Completed	
04 – Mini Remote Sensing Task (NetCDF + xarray)	Completed	
05 – Capstone Assignment (01–04)	Completed	

2.2 – Short Summary (5–8 sentences)

In your own words, describe what this Python component of the course covered, and how it relates to remote sensing / ocean / atmosphere / climate applications.

In the python coarse we covered the components such as about variables, about earth-science calucaltion, data types such as- int, float, str and bool and also using the type() to check it and even studied to convert it and print the values. We also studied on topics of temperature conversion, calculating basic math & statistics in pyhton, arithmetic. Learnt to create list, loops, classify the data, about Numpy, NetCDF, Xarray, Matplotlib, inline, function, about anomalies, plotting of data, dataset structure and how to access it. All this information hepled in creating variables and munipulating it, using print() to showcase basic expressions, give data types, to interpret the errors, to calculate earth/atmos/ocean-related calculations.

---

3. Notebook 01 – Python Fundamentals (Summary)

3.1 – Key Concepts Learned

Write 4–6 bullet points summarising what you learned in Notebook 01 (variables, operators, basic statistics, etc.).

- Varaibles is a name that stores a memory of the value.
- Operators are symbols or keywords commands given to python to perform that task. eg: add, subtract, divide, etc.
- A comment is text that Python ignores.It starts with # and is used to explain what your code is doing.
- Correlation tells us how strongly two variables move together
- Code cells where you write and run Python.

3.2 – Example Code Snippet

Paste a **small piece of code** from Notebook 01 that you are proud of or found useful (for example: mean/median/std computation).
You can re-type or copy-paste and then **run** it here.

```
# Example code from Notebook 01 - edit this cell with your code
import statistics as stats
sst = [28.1, 28.3, 28.3, 27.9, 28.5, 28.7, 28.3]

mean_sst = stats.mean(sst)
median_sst = stats.median(sst)
mode_sst = stats.mode(sst)

print("Mean SST:", mean_sst)
print("Median SST:", median_sst)
print("Mode SST:", mode_sst)
```

```
Mean SST: 28.3
Median SST: 28.3
Mode SST: 28.3
```

3.3 – Short Reflection (3–4 sentences)

What part of basic Python felt easiest and what part required practice?

The use of print function after sometime felt easy compared to before. Also the computation of addition, subtraction, division, multiplication of values felt easier. But on how to print it using what function required some time to understand it.

---

4. Notebook 02 – Lists, Loops, and Conditions (Summary)

4.1 – Key Concepts Learned

List 4–6 points about lists, loops, and if-else conditions that you learned.

- a list is an ordered collection of items.
- a loop is a control structure used to repeat a block of code multiple times.
- a condition is a logical expression that determines whether a specific segment of code should execute.
- if-else conditions are decision-making tools that evaluate a logical expression; if the condition is True, the code block under if executes, otherwise, the program runs the alternative code block under else.
- filtering is the process of selecting specific elements from a collection (like a list) that meet a certain condition.

4.2 – Example: SST Classification

Using a short SST list, re-create your classification of days (HOT / WARM / NORMAL) here.

```
# Recreate your SST classification logic here

sst = [28.1, 28.3, 28.4, 28.0, 27.9, 28.5, 28.7, 28.6]

# Write your classification code
hot_sst = []
normal_sst = []
threshold = 28.5

for i in range(len(sst)):
    temp = sst[i]
    if temp > threshold:
        hot_sst.append(temp)
    if temp < threshold:
        normal_sst.append(temp)

print("Hot SST values:", hot_sst)
print("Normal SST values:", normal_sst)
```

```
Hot SST values: [28.7, 28.6]
Normal SST values: [28.1, 28.3, 28.4, 28.0, 27.9]
```

4.3 – Reflection (3–4 sentences)

Why are loops and conditions important in scientific data analysis?

In scientific data analysis, loops are essential for automating repetitive tasks across massive datasets—such as processing millions of sensor readings or genomic sequences—enabling researchers to perform complex calculations and iterative simulations that would be impossible manually. Conditions provide the "intelligence" needed for data quality control, allowing programs to automatically filter out noise, handle missing values, and execute different analytical logic based on specific experimental thresholds or criteria. Together, they transform raw, unstructured measurements into refined results by applying consistent, rule-based logic at a scale required by modern 2025 research standards.

---

5. Notebook 03 – Functions and Plotting (Summary)

5.1 – Key Concepts Learned

Write 4–6 bullet points on functions, NumPy arrays, and plotting.

- Functions are used to wrap repetitive data processing steps into reusable blocks, which reduces errors and allows researchers to apply the same analytical logic across different experimental trials or datasets.
- NumPy arrays (specifically the ndarray) store data of the same type in contiguous memory blocks, enabling "vectorized" operations that run at compiled C speed—essential for handling millions of data points efficiently.
- NumPy allows you to perform mathematical operations (like array * 2) on entire datasets at once without writing slow for loops, significantly reducing execution time for complex scientific simulations and statistical calculations.
- Matplotlib remains the standard for creating highly customizable, publication-quality static plots, while modern libraries like Plotly

- NumPy arrays serve as the fundamental data format that bridges raw data handling in Pandas with advanced modeling in SciPy and professional visualization in Matplotlib or Seaborn.

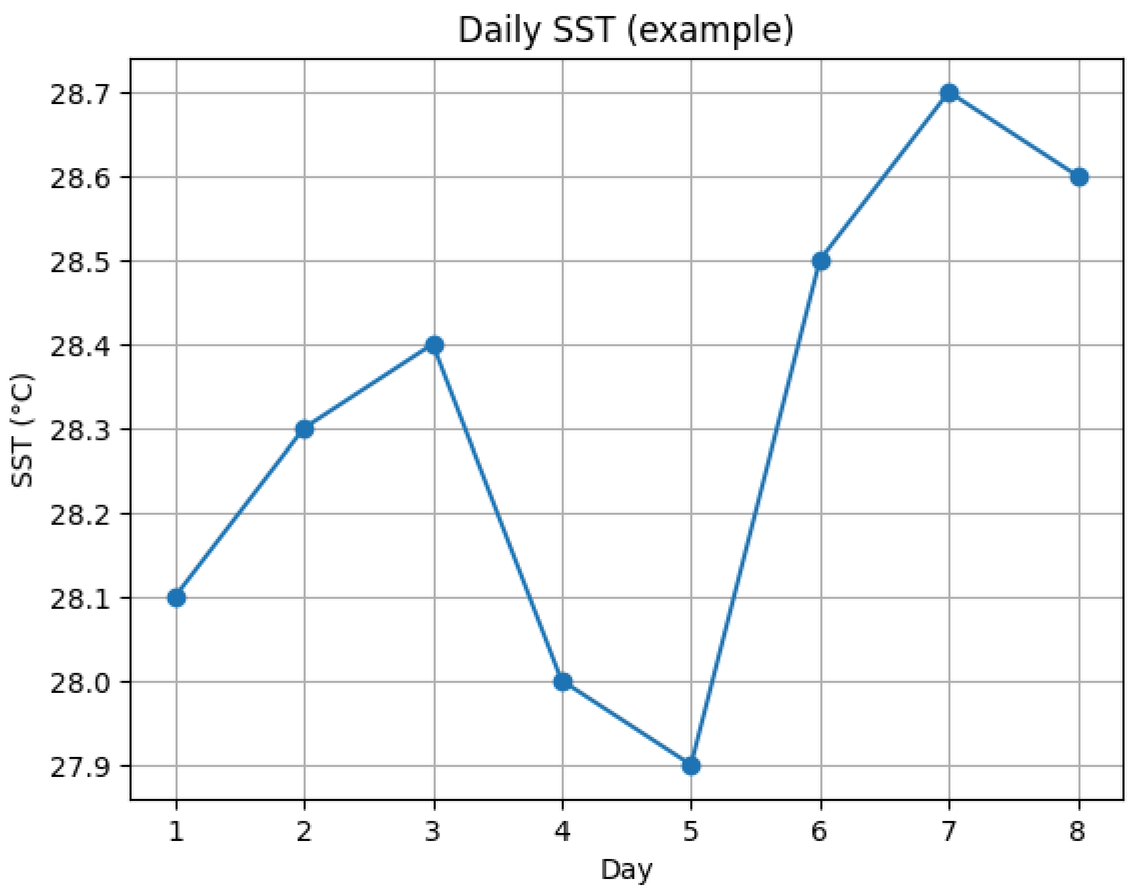
5.2 – Plot Re-creation: SST and Anomalies

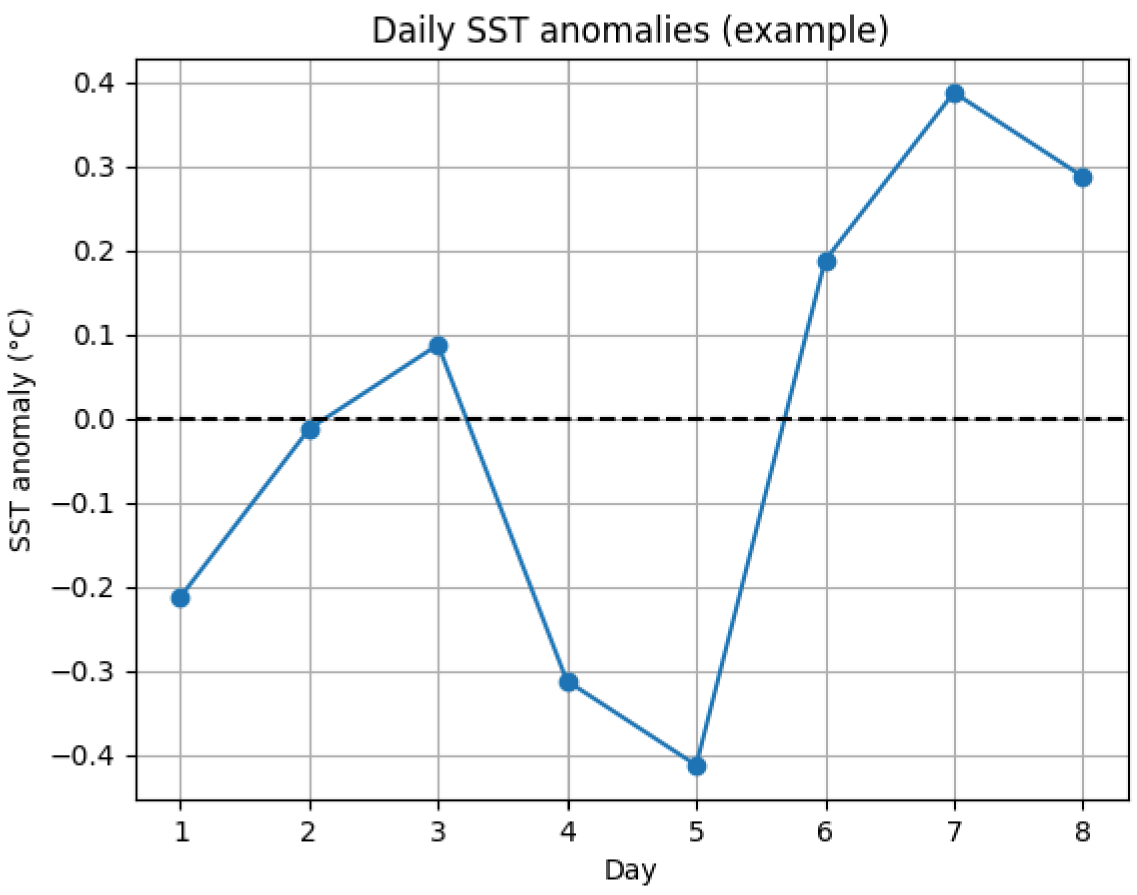
Below, reproduce **two plots**:

1. SST vs. day
2. SST anomaly vs. day (with a horizontal zero line)

Use a small SST example (you may reuse one from previous notebooks).

```
# Recreate SST and anomaly plots here

import numpy as np
import matplotlib.pyplot as plt

# Example SST data (edit as needed)
sst = np.array([28.1, 28.3, 28.4, 28.0, 27.9, 28.5, 28.7, 28.6])
days = np.arange(1, len(sst)+1)

# Compute anomalies
mean_sst = sst.mean()
sst_anom = sst - mean_sst

# Plot 1: SST
plt.figure()
plt.plot(days, sst, marker='o')
plt.xlabel("Day")
plt.ylabel("SST (°C)")
plt.title("Daily SST (example)")
plt.grid(True)
plt.show()

# Plot 2: Anomalies
plt.figure()
plt.plot(days, sst_anom, marker='o')
plt.axhline(0, color='black', linestyle='--')
plt.xlabel("Day")
plt.ylabel("SST anomaly (°C)")
plt.title("Daily SST anomalies (example)")
plt.grid(True)
plt.show()
```





5.3 – Reflection (3–4 sentences)

How did plotting help you understand the behaviour of SST and its anomalies?

Plotting the Sea Surface Temperature (SST) data provided immediate clarity into its behavior and anomalies, revealing patterns far more quickly than numerical data alone.

Key Visual Understandings

Instantaneous Trend Analysis: The line graph quickly communicated the fluctuating nature of the SST over the eight days, showing periods of warming and cooling that defined the overall trend.

Identification of Extremes: The plot made it easy to pinpoint the most significant anomalies, clearly marking the warmest point (28.7°C) and coldest point (27.9°C) relative to the mean of 28.3125°C.

Understanding Variability: The steep lines in the graph visually represented the rate of change between days, indicating rapid shifts in temperature that signify high local variability.

Intuitive Comparison: The visual format allowed for quick comparison of each day's temperature against the others and the implied average, making the magnitude and direction of each anomaly instantly obvious.

---

6. Notebook 04 – NetCDF & xarray Remote Sensing Task (Summary)

6.1 – Dataset Description

Describe the sample NetCDF dataset you worked with:

- Variable name(s):
- Dimensions (e.g. time, lat, lon):
- Units:
- Approximate region covered:

<xarray.Dataset> Size: 7kB

Dimensions: (time: 7, lat: 11, lon: 11) Coordinates:

- time (time) datetime64[ns] 56B 2020-01-01 2020-01-02 ... 2020-01-07
- lat (lat) float64 88B 10.0 11.0 12.0 13.0 14.0 ... 17.0 18.0 19.0 20.0
- lon (lon) float64 88B 70.0 71.0 72.0 73.0 74.0 ... 77.0 78.0 79.0 80.0

Data variables: sst (time, lat, lon) float64 7kB ...

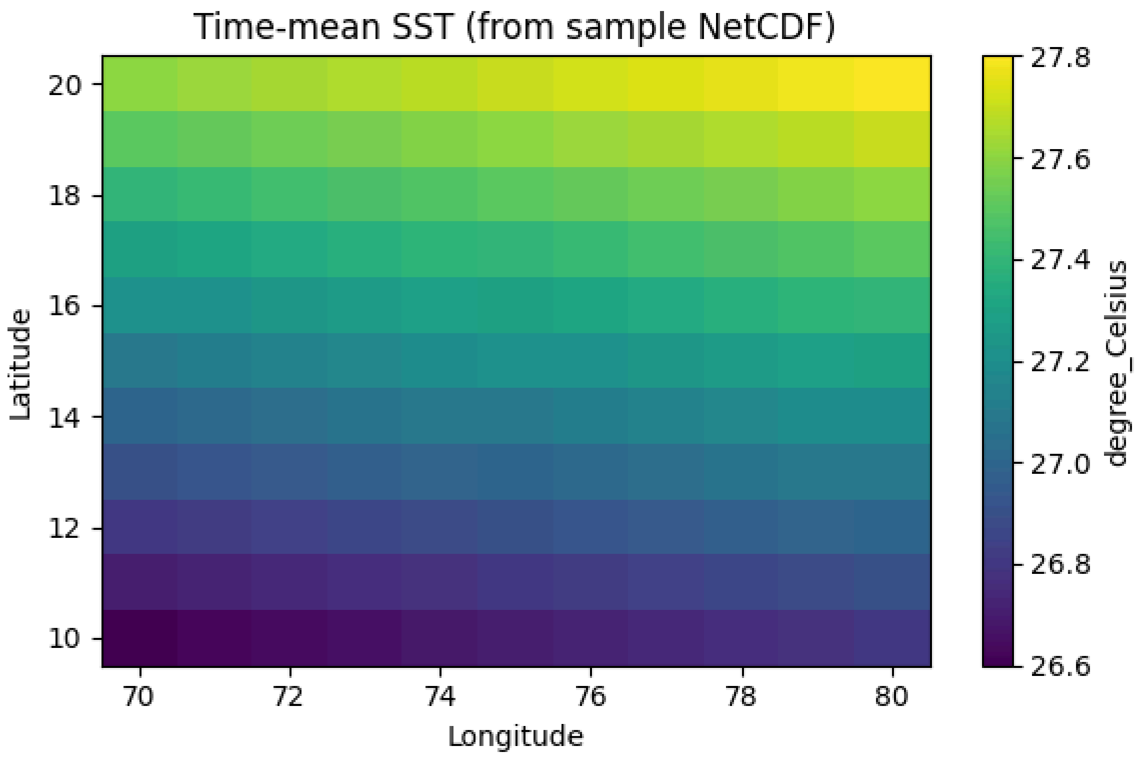
Attributes: title: Sample synthetic SST dataset for teaching source: Generated in Python for teaching by Arjun's EarthSystem-Science..

6.2 – Recreate Time-Mean SST Map

Run the code below to recompute and plot the time-mean SST map.

> Note: This cell auto-downloads the `data_sst.nc` file from the GitHub repository.

```
import os, requests
import xarray as xr
import matplotlib.pyplot as plt

# Download NetCDF file if not present
url = "https://raw.githubusercontent.com/EarthSystem-Science-Lab/python-basi
local = "data_sst.nc"

if not os.path.exists(local):
    r = requests.get(url)
    open(local, "wb").write(r.content)

ds = xr.open_dataset(local)
sst = ds["sst"]

# Compute time-mean
sst_mean_time = sst.mean(dim="time")

plt.figure(figsize=(6,4))
plt.pcolormesh(ds["lon"], ds["lat"], sst_mean_time, shading="auto")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Time-mean SST (from sample NetCDF)")
cbar = plt.colorbar()
cbar.set_label(sst.attrs.get("units",""))
plt.tight_layout()
plt.show()
```

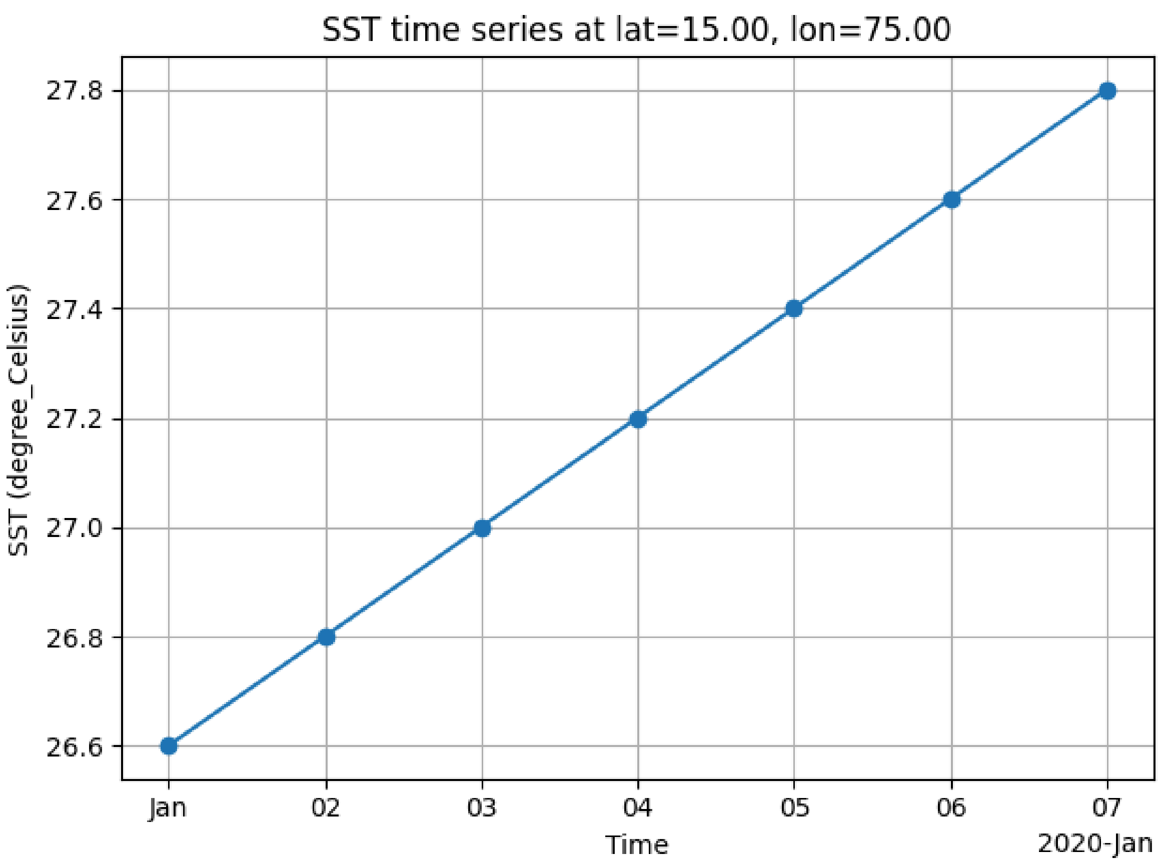


6.3 – Recreate SST Time Series at a Point

Choose a latitude and longitude inside the dataset domain and plot the time series.

```
import os, requests
import xarray as xr
import matplotlib.pyplot as plt

# Download NetCDF file if not present
url = "https://raw.githubusercontent.com/EarthSystem-Science-Lab/python-basi
local = "data_sst.nc"

if not os.path.exists(local):
    r = requests.get(url)
    open(local, "wb").write(r.content)

ds = xr.open_dataset(local)
sst = ds["sst"]

# Choose a point (edit values if you like)
lat_pt = float(ds.lat.sel(lat=ds.lat.mean(), method="nearest"))
lon_pt = float(ds.lon.sel(lon=ds.lon.mean(), method="nearest"))

ts = sst.sel(lat=lat_pt, lon=lon_pt, method="nearest")

plt.figure()
ts.plot(marker='o')
plt.title(f"SST time series at lat={lat_pt:.2f}, lon={lon_pt:.2f}")
plt.xlabel("Time")
plt.ylabel(f"SST ({sst.attrs.get('units','')})")
```

```
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```



6.4 – Reflection (4–6 sentences)

Describe what you learned about:

- NetCDF as a format
- xarray usage
- Basic remote sensing data handling in Python

- NetCDF (Network Common Data Form) is a highly prevalent, self-describing, and machine-independent file format tailored for storing complex, multidimensional scientific data like that found in climate modeling and oceanography. Its key strength lies in its ability to embed all necessary metadata—dimensions, units, variable descriptions—within the file itself, ensuring that any researcher or program can understand the data without external documentation. Furthermore, modern NetCDF-4 leverages the robust HDF5 standard internally, offering advanced features such as data compression and hierarchical grouping which are essential for managing the sheer scale of modern Earth science data efficiently across different platforms.
- Xarray is a critical Python library designed to bridge the gap between simple numerical arrays and complex, real-world scientific data. It introduces "labeled

indexing" by applying the powerful coordinate-based selection capabilities of the pandas library to multidimensional NumPy arrays. This means researchers can select data using intuitive labels (like 'time', 'latitude', or specific dates) rather than confusing integer indices, making analysis code significantly more readable and robust. Xarray also serves as the de facto standard for reading, manipulating, and writing NetCDF files in the Python ecosystem, integrating seamlessly with Dask for out-of-core and parallel processing of datasets that exceed available memory.
- Basic remote sensing data handling in Python typically involves a workflow built upon these core tools. Researchers use specialized clients, often leveraging the STAC standard, to efficiently search for satellite imagery in cloud archives. Once data is accessed (frequently as NetCDF or GeoTIFF files handled by Xarray and its geospatial extension, rioxarray), the focus shifts to data cleaning and preparation. This crucial step involves applying scale factors to raw sensor data and using conditional logic to perform rigorous cloud masking, filtering out unreliable pixels using quality assurance bands. The final step involves computing spectral indices (like NDVI) using fast, vectorized operations and visualizing the results with libraries like Matplotlib or Plotly.

---

7. Capstone Assignment (Summary)

If you completed the **Capstone Assignment**, summarise them here.

In the Capstone Assignment basic arithmetic and varaibles were computed. Basic statistics such as mean, median, minimum, maximmum, standard deviation were also calculated. Day-wise SST were printed, classified into HOT, WARM and NORMAL based on the days, a rolling 3-day mean was also computed. Funtion such as Celsius → Kelvin, Anomaly Computation were done. Dataset was explored, conversion of Numpy to plot, statistics, Time-mean map, Time series at a location were also computed.

7.1 – Capstone Assignment

In 6–10 sentences, describe:

- The overall flow of the capstone assignment
- How it connected Python skills to a realistic remote sensing / Earth system problem
- What you found most interesting
- Which part you would like to explore further

In the Capstone Assignment basic arithmetic and varaibles were computed. Basic statistics such as mean, median, minimum, maximmum, standard deviation were also calculated. Day-wise SST were printed, classified into HOT, WARM and NORMAL based on the days, a rolling 3-day mean was also computed. Funtion such as Celsius → Kelvin, Anomaly Computation were done. Dataset was explored, conversion of Numpy to plot, statistics, Time-mean map, Time series at a location were also computed. The project connects Python skills to real-world issues by using open-source libraries (Xarray, NumPy, rioxarray) to automate tasks that were once performed manually or required expensive desktop GIS software. For example, the skills were applied to process massive datasets of satellite imagery (remote sensing) to monitor Earth systems, such as identifying areas of vegetation stress (using NDVI) or mapping changes in water bodies, providing tangible evidence for environmental investigations. A highly interesting part is leveraging libraries like Xarray with Dask integration, which allows the analysis of datasets far larger than the computer's memory by processing data in parallel. This ability to handle "big data" efficiently in a standard Python environment makes complex, planetary-scale analysis accessible and efficient for standard research. The most valuable area for further exploration would be advanced cloud-native data access and machine learning integration. Using Google Earth Engine or similar platforms in combination with Python allows for rapid analysis without downloading terabytes of data, and integrating machine learning algorithms (using scikit-learn) could automate complex tasks like land cover classification with greater accuracy.

---

8. Final Reflection on Python & Remote Sensing

Write 8–12 sentences reflecting on the **entire Python component** of this course.

You may cover:

- How your comfort with Python has changed
- Which concepts (lists, loops, functions, plotting, xarray) you feel confident about
- Where you still need practice
- How you plan to use these skills in future courses, projects, or research
- Any ideas you have for applying Python to real datasets from ocean, atmosphere, climate, or land surface studies

The Python component of this course significantly transformed my comfort level with programming, moving me from basic syntax understanding to functional data analysis capabilities. I now feel highly confident in using fundamental Python concepts such as lists, loops, and functions to structure robust and efficient code. My comfort extends to utilizing NumPy arrays for high-performance numerical operations and generating effective data visualizations using plotting libraries like Matplotlib. I also feel capable of basic remote sensing data handling using Xarray to open and explore NetCDF files. However, I still recognize a need for more practice in advanced geospatial operations, particularly in managing coordinate reference systems and mastering complex Dask integration for truly "big data" parallel computing. I plan to leverage these skills extensively in future courses and personal projects, aiming to apply Python to real-world atmospheric and oceanographic datasets. Specifically, I intend to build a project focused on analyzing global sea surface temperature (SST) anomalies using satellite data or modeling localized climate impacts on land surface hydrology.

---

9. Declaration & Signature

I, **Vedika Kavlekar**, declare that this report is based on my own work and understanding.
Where I have used external resources or received help, I have acknowledged it