How to Handle Large Datasets?

Group No. 11

Rayyan Hassan  
2021535

Faakhir Inam   
2021152

Sibtain Haider  
2021459

Haseeb Ahmer  
2021903

*Abstract*— The use of Principal Component Analysis (PCA) to solve issues with managing huge geographic datasets is examined in this paper. The work highlights error analysis and ideal parameter selection through data visualization, picture cropping, and PCA application. [1].

# Introduction

Large datasets provide difficulties for efficient analysis in many domains. To improve computing efficiency, this paper looks into the application of PCA for geospatial data, with a particular emphasis on dimensionality reduction. The purpose of this report's structure is to thoroughly examine PCA implementation of geospatial data. Every part helps to reveal how PCA may be used to overcome the difficulties presented by big datasets, from the first phases of data visualization and preprocessing to the complexities of eigenvalue decomposition and dimensionality reduction. The methodology, findings, and analysis covered in the ensuing sections offer a comprehensive assessment of the application's effectiveness. Our goal is to further the current conversation on scalable and effective geospatial data analysis, opening the door to a more profound understanding of our dynamic environment.

# Libraries Installed

To make geospatial data processing easier, the necessary Python libraries were installed, such as matplotlib and rasterio.

## Rasterio

To see the composition of the chosen Sentinel-2 image, it is opened and shown in several bands. A popular and robust Python library for reading, writing, and modifying geographic raster data is called Rasterio. Rasterio plays a crucial role in the data preprocessing pipeline for this project because the input data is raster images. Its capabilities include quickly opening geographical datasets, obtaining pixel values, and supplying crucial metadata that serves as the basis for further analysis[2]. Rasterio removes the need for preprocessing procedures by smoothly integrating with raster data formats like JP2 (JPEG 2000), freeing users to concentrate on the essential elements of geographic analysis. The data is easily made to be a 2D vector array containing data in the format that we prefer.

## MatplotLib

Matplotlib is a flexible charting package that is frequently used to generate interactive, animated, and static Python visualizations [3]. When it comes to visually inspecting the input raster pictures, investigating the distribution of pixel values over different bands, and displaying the results of various processing stages, Matplotlib proves to be an indispensable tool in the context of geographic data analysis. Raster data can be shown using show functions, which makes it easier to spot spatial patterns. An example of MATLAB code that was utilized in this CEP is shown below. Three distinct bands are displayed in a subplot as the output.

plt.subplot(3, 4, i + 1)

plt.imshow(stacked\_data[:, :, i])

plt.title(f'Band {i + 1}')

plt.axis('off')

## Numpy

NumPy is an essential Python library for scientific computing that supports big, multi-dimensional matrices and arrays as well as mathematical operations on them. One application of NumPy is the computation of eigenvalues and eigenvectors, which are necessary for PCA implementation.

eigenvalues,eigenvectors=np.linalg.eig(covariance\_matrix)

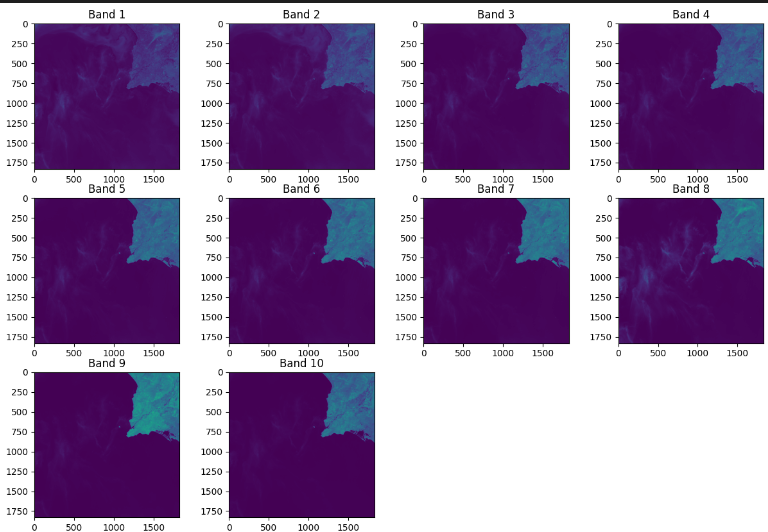
## Scikit

This library is used to calculate the mean square error which is a method used to calculate the optimal number of Eigen Vectors to be used.

# basic operations

## Image Visualozation

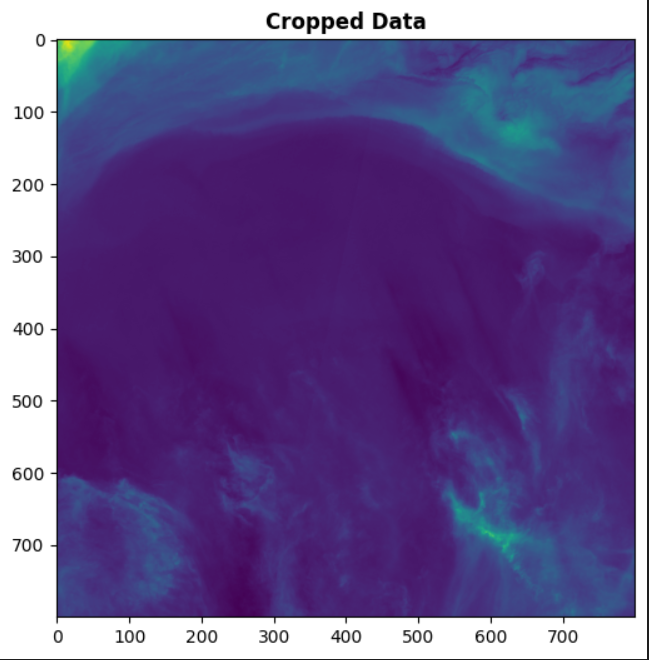
To comprehend its composition, the chosen Sentinel-2 image is opened and shown in various bands[4]. This picture is intricate. When it comes to deciphering the complexities of geographic datasets, image visualization is essential because it provides a visual story that improves comprehension and makes it easier to spot patterns and anomalies. The geographical dataset's many bands are visualized using subplots made using Matplotlib. Each subplot represents a separate spectral band.



The first band of this image illustrates that we can identify features in raw numerical information that may be obscured by the visual depiction of geospatial data. For example, variances in color intensity between bands may be a sign of different vegetation health, land cover, or urban growth.  *It can be seen that Band 10 shows less water vapour as compared to the other bands.*

## Image Cropping

The cropping of the image demonstrates the capacity to process specified portions within enormous datasets by highlighting a particular location of interest. The first step in creating an image is to define an area of interest (ROI). This area has been carefully chosen by the analysis's goals. It could be any target of interest, such as a location with land cover characteristics or one that is vulnerable to environmental changes. After determining the image's width and height, we can focus on a particular region to capture a photo of it.



cropped\_image=stacked\_data[:800, :800, :]

plt.figure(figsize=(6, 6))

show (cropped\_image[:,:, 0], title='Cropped Data')

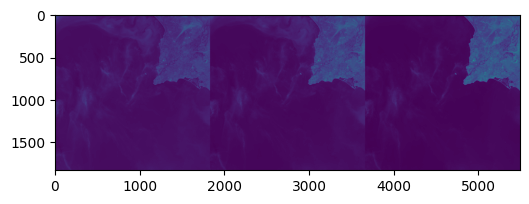
## Concatenate Bands

To create a single composite image for analysis and display, multiple bands from a multispectral image are combined. This process is known as image concatenation. In this technique, a uniform representation is produced by concatenating the bands of the Sentinel-2 image. Through this approach, we may jointly examine and process the complete multispectral dataset, leading to a more thorough knowledge of the spectral and spatial data.

concatenated\_image= np.concatenate((stacked\_data[:, :, 0], stacked\_data[:, :, 1], stacked\_data[:, :, 2]), axis=1)

plt.figure(figsize=(8, 8))

show(concatenated\_image)



# PCA applying technique

## Reshaping and Concatenation

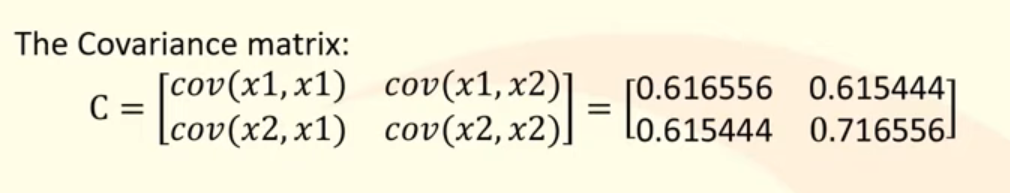
The geospatial data must be reshaped into an analysis-ready format before using PCA[5]. To do this, the data are frequently flattened, turning a multi-dimensional array into a two-dimensional representation. This reshaping is essential in the context of raster data to convert the image with numerous bands and spatial dimensions into a format where each row and each column represent a band and a pixel, respectively.

flattened\_data=stacked\_data.reshape((-

1, stacked\_data.shape[-1])))

## Covariance Matrix Calculation

When the covariance matrix is calculated, the relationships between various bands become clear. The covariance matrix, which in this case represents the correlations between several variables (bands), is calculated as a prerequisite for PCA[5]. The covariance matrix sheds light on the differences between each band and the others. Comprehending the interdependencies and redundancies present in the dataset requires completing this crucial stage. An illustration of a smaller dataset with x1 and x2 values may be found in the image below.



We are to calculate the mean values of the flattened data, and subtract it from the flattened data to achieve a more precise dataset. This will make our calculations of eigenvalues and vectors easier. Not centralizing our data may lead to results such as below.

A graph with blue squares

Description automatically generated

## Eigenvalue and Eigenvector calculations

## 

## 

*Derivation of eigenvalues and eigenvectors from the covariance matrix Eigenvectors show the directions of maximal variance, while eigenvalues show the variance of the data along the principal components. Finding the primary components depends on these eigenvalues and eigenvectors. The eigenvectors are arranged in descending order according to their corresponding eigenvalues to identify the principal components that capture the greatest variance[5].*

sorted\_indices=np.argsort(eigenvalues)[::-1]

sorted\_eigenvectors = eigenvectors[:, sorted\_indices]

## Dimensionality Reduction

Selecting the top-k eigenvectors to keep in the analysis is the last step. The desired degree of dimensionality reduction is frequently taken into consideration while choosing 'k'. A lower 'k' indicates a more significant decrease, but there may be some information loss.

k = 10

selected\_eigenvectors=sorted\_eigenvectors[:, :k]

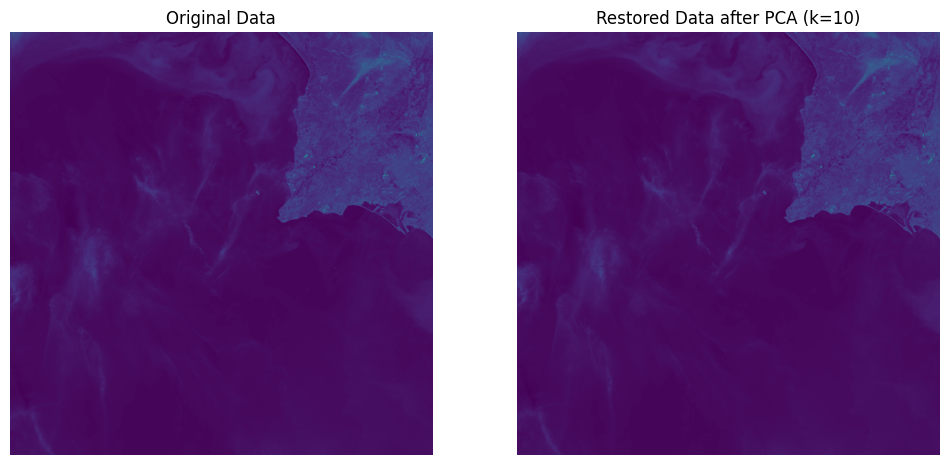
The value of K needs to be a compromise between image complexity and accuracy.

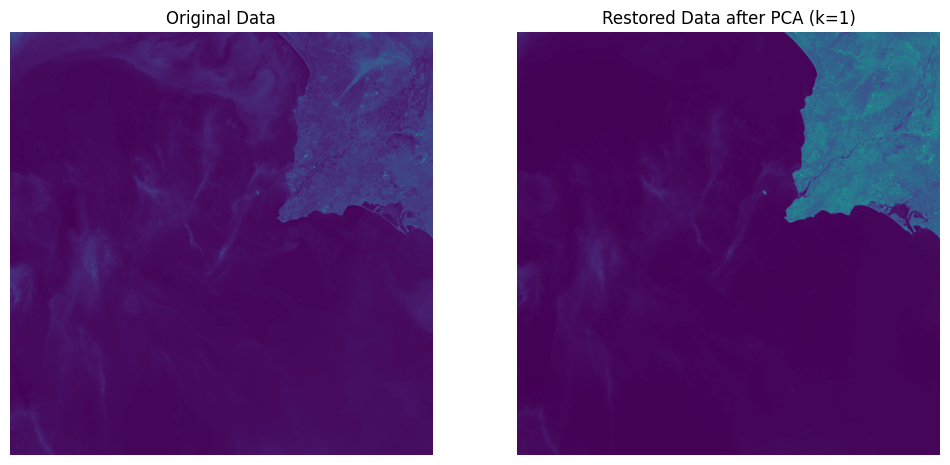
## Restoration and Visualization

The original image is then restored using the reduced data that is obtained by multiplying the chosen eigenvectors with the original centered data, as the image illustrates. The effect of dimensionality reduction is then evaluated by visualizing the restored data alongside the original data.

## A screenshot of a calculator Description automatically generated

The image is different for unique values of k.





# Error analysis

Analysis is done on reconstruction errors (RMSE) for various values. The ideal number of principal components (k) to use in error analysis is one of the main factors to take into account. The relationship between model complexity and accuracy is influenced by the value of 'k'. A thorough examination of a variety of 'k' values is frequently conducted to determine the point at which the dimensionality reduction has the least negative effect on the reconstruction's correctness. Each case's Root Mean Squared Error (RMSE) is computed after testing a range of 'k' values. This is the starting point for figuring out the ideal "k" that strikes a balance between accuracy and complexity. The value of k cannot be more than the number of bands. In this case, the number of bands used is ten. The value of k needs to be less than that. To choose which value to utilize, we will now experiment with values close to ten. We are going to compute the RMS for every k value in that vicinity.

for k in range(1, max\_k + 1):

# Choose the top k eigenvectors

selected\_eigenvectors = sorted\_eigenvectors[:,: k]

# Project the data onto the selected eigenvectors

reduced\_data = np.dot(centered\_data, selected\_eigenvectors)

# Reconstruct the data

reconstructed\_data = reconstruct\_data(reduced\_data, selected\_eigenvectors)

try:

# Calculate MSE and store the result

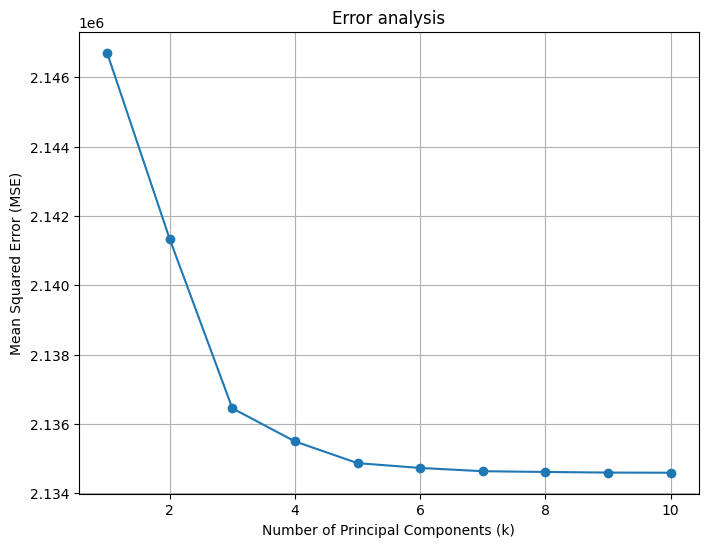
mse = mean\_squared\_error(flattened\_data, reconstructed\_data)

mse\_values.append(mse)

except ValueError as e:

print(f"Error for k={k}: {e}")

we again use the dot product of the centered data and eigenvectors to reconstruct the data. We calculate the mean squared error. Because Mean Squared Error (MSE) is a useful metric for measuring predictive model accuracy, it is frequently used in a variety of domains, such as statistics, machine learning, and signal processing. MSE measures the average squared difference between expected values and actual observations. MSE penalizes larger errors more severely than smaller ones by squaring the differences. Because of this feature, MSE is especially appropriate for applications where it is important to minimize the impact of outliers. MSE offers a thorough assessment of the overall performance of the model in the context of model evaluation. Its simple interpretation as the average squared error makes comparing various models or parameter configurations easier. Furthermore, MSE's mathematical characteristics—such as its sensitivity to variations from the true values and non-negativity—make it a reliable tool for evaluating goodness of fit. Additionally, MSE is frequently used in optimization problems where the goal is to reduce prediction errors. Because it is consistent with the idea of minimizing the expected value of the squared error, it is used in these situations. Given the circumstances, MSE's interpretability, simplicity, and adaptability have led to its broad acceptance as a trustworthy metric for assessing and improving models in a variety of fields [6].



# task distribution

Everyone did the project together. The research work was divided and explained to one another. Once we obtained an understanding of the PCA techniques we moved forward in writing the code. All of us had an idea of Python hence we used stack exchange and YouTube tutorials to make a code together. Regardless these are the major roles played by everyone.

## Haseeb Ahmer

Haseeb Ahmer contributed by doing research regarding what libraries we may use for our codes. He recommended the rasterio library to open the picture. Moreover, he performed basic operations such as visualizing different bands and cropping the image.

## Rayyan Hassan

Rayyan was responsible for downloading the initial SAFE file from the website. He wrote the code for the PCA applying technique and oversaw debugging all the other code to ensure a smooth project.

## Fakhir Inam

Faakhir was responsible for Stacking the image that was to be used in our calculations. He made sure the dataset was ideal and filled all prerequisites for PCA applying techniques. Faakhir was also responsible for drafting the report, using the IEEE template.

## Sibtain Haider

Sibtain was responsible for writing the error analysis part of the code, he recommended the optimal number of eigenvectors to be used. Sibtain also was responsible for proofreading the Report, adding pictures and references where necessary.

##### References

1. https://www.researchgate.net/publication/372252738\_Assessing\_the\_Dimensionality\_Reduction\_of\_the\_Geospatial\_Dataset\_Using\_Principal\_Component\_Analysis\_PCA\_and\_Its\_Impact\_on\_the\_Accuracy\_and\_Performance\_of\_Ensembled\_and\_Non-ensembled\_Algorithms
2. https://gis.stackexchange.com/questions/397638/opening-sentinel-2-data-with-python
3. https://www.w3schools.com/python/matplotlib\_pyplot.asp
4. https://dataspace.copernicus.euR
5. https://youtube.com/playlist?list=PL\_jTZi-bjHgTmTBuqFB-9DK0ka8eASGl-&si=4j6fypTU0aC99ifV
6. https://neustan.wordpress.com/tag/mean-squared-error/#:~:text=PCA%20simply%20finds%20the%20plane,of%20the%20shortest%20orthogonal%20distance.