# Session 1: Copernicus Sentinel Data Deep Dive & Philippine EO Ecosystem

**Overview:** This session introduces the European Copernicus Earth Observation program with a focus on the Sentinel-1 and Sentinel-2 missions, and reviews key Earth Observation (EO) stakeholders and data resources in the Philippines. Instructors should emphasize the characteristics of Sentinel data (spatial/spectral/temporal resolution, data products) and how local Philippine agencies and platforms can complement these datasets. An activity at the end will familiarize participants with the CopPhil data **Mirror Site** and the **Digital Space Campus** for accessing EO data and training materials.

## Copernicus Program and Sentinel Missions

**Copernicus Program:** Begin by outlining the Copernicus program as the European Union’s flagship Earth Observation initiative providing **free and open satellite data**. Highlight that Copernicus comprises a fleet of **Sentinel satellites** designed for environmental monitoring. Stress the importance of this program for global and local applications (disaster management, climate, resource monitoring). You can cite that Copernicus offers a *“multi-petabyte catalog of satellite imagery and geospatial datasets with planetary-scale analysis capabilities”*[[1]](https://earthengine.google.com/#:~:text=Google%20Earth%20Engine%20combines%20a,for%20academic%20and%20research%20use) available for scientists and practitioners worldwide.

**Sentinel-1 (SAR):** Describe Sentinel-1 as a **synthetic aperture radar (SAR)** mission (radar imagery) with **C-band** sensors. Key points to mention: it provides all-weather, day-and-night observations unaffected by clouds, which is crucial in tropical regions. Sentinel-1 operates in specific modes (the default **Interferometric Wide (IW)** mode has a spatial resolution of about **5 m by 20 m** in range and azimuth[[2]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=focusing%20on%20the%20Sentinel,Engine%2C%20will%20be%20thoroughly%20explained)). It has a **6-12 day revisit** cycle globally with a two-satellite constellation (Sentinel-1A and 1B), though note for the instructor: Sentinel-1B became inoperative in 2022 – Sentinel-1C is slated to join and restore the 6-day revisit in the future (mention if relevant). Explain that Sentinel-1 data comes as **Level-1 GRD (Ground Range Detected)** products (detected, multi-looked SAR images projected to ground range) most commonly, and also **SLC (Single Look Complex)** for advanced processing like interferometry. As an instructor note, you may show an example Sentinel-1 image, pointing out how water vs. urban areas appear in radar imagery (if a visual is available). Emphasize that SAR data requires interpretation (speckle noise, polarization – VH and VV polarizations for Sentinel-1 IW mode over land).

**Sentinel-2 (Optical):** Sentinel-2 is an **optical multispectral** mission with a focus on land imaging. Each Sentinel-2 satellite carries the **Multi-Spectral Instrument (MSI)** with **13 spectral bands** spanning visible, near-infrared, and shortwave-infrared wavelengths. Describe the spatial resolution tiers: **4 bands at 10 m** (Blue, Green, Red, Near-IR), **6 bands at 20 m** (red-edge and SWIR bands), and **3 bands at 60 m** (for atmospheric correction, e.g. aerosol, water vapor, cirrus)[[2]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=focusing%20on%20the%20Sentinel,Engine%2C%20will%20be%20thoroughly%20explained). The constellation of two satellites (2A and 2B) provides a revisit of **5 days** at the equator (10 days per satellite)[[2]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=focusing%20on%20the%20Sentinel,Engine%2C%20will%20be%20thoroughly%20explained). This high revisit is great for monitoring changes and getting cloud-free views by compositing images. Mention standard Sentinel-2 data products: **Level-1C** (Top-of-Atmosphere reflectance, 10/20/60 m, in tiling grid) and **Level-2A** (Bottom-of-Atmosphere reflectance i.e. surface reflectance, with cloud masks, available operationally since late 2018)[[2]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=focusing%20on%20the%20Sentinel,Engine%2C%20will%20be%20thoroughly%20explained). Note to instructors: explain that Level-2A includes the Scene Classification Layer (SCL) which labels pixels as cloud, shadow, vegetation, etc., which can be used for cloud masking in later exercises.

**Data Access Methods:** Explain how to obtain Sentinel data. European **Copernicus Open Access Hubs** (now transitioned to the **Copernicus Data Space Ecosystem**) allow direct downloading of Sentinel-1 and 2 scenes. More conveniently for this training, **Google Earth Engine (GEE)** provides ready **data catalogs** for Sentinel-1 (GRD) and Sentinel-2 (Level-2A) – we will use GEE extensively for data access. Mention that **Copernicus Hub** services were phased out in 2023 in favor of the new Data Space platform[[3]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=stakeholders). As an instructor, clarify that all Sentinel data are free and open; participants can later explore other sources like AWS registry, Sentinel Hub, or national mirror sites. It’s useful to cite that Copernicus data supports many applications *“in areas such as emergency management, health care, agriculture, environment, with crucial impact on society, climate, and economy”*[[4]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=Copernicus%20is%20the%20European%20Earth,applications%20based%20on%20satellite%20data) to convey the importance of accessing and using this data.

*Teaching Note:* You might show a short demo of browsing the Copernicus Data Space or the ESA SciHub (if still accessible) just to illustrate how one would search for a Sentinel-2 tile by date/location. However, since we will do data access via GEE, keep this part brief. Emphasize to participants that they won’t need to manually download Sentinel imagery during the training – instead, they will leverage cloud platforms.

## The Philippine EO Landscape

Now shift focus to the **national context in the Philippines**. Participants should recognize local institutions and data sources that complement the Copernicus data. Each of the following agencies provides datasets or support that can enhance AI/ML projects in EO:

* **Philippine Space Agency (PhilSA):** Introduce PhilSA as the central civilian space agency of the Philippines (established 2019). PhilSA is a key player in promoting EO data use. It operates platforms like the **Space Data Dashboard (Space+)** – an online portal where users can browse satellite images and products relevant to the Philippines. PhilSA is also hosting the upcoming Copernicus mirror data center. Highlight that PhilSA is a co-chair of this EU-funded CopPhil program, indicating strong support for building local EO and AI capacity.
* **National Mapping and Resource Information Authority (NAMRIA):** NAMRIA is the national mapping agency, responsible for topographic maps, hydrographic data, and more. They maintain a **Geoportal** that offers access to base maps, administrative boundaries, and thematic maps (e.g., land cover, hazard maps). Mention that the **NAMRIA Geoportal** can be a source of reference data or ground truth for validation – for example, official land cover maps or flood hazard maps that could be used alongside satellite data. As a teaching note, if internet permits, you could briefly show the NAMRIA Geoportal interface or describe how one can download datasets from it (e.g., shapefiles of administrative boundaries or land cover).
* **Department of Science and Technology – Advanced Science and Technology Institute (DOST-ASTI):** DOST-ASTI leads several projects at the intersection of remote sensing, AI, and big data:
* **DATOS** (Remote Sensing and Data Science Help Desk): Explain that DATOS was a program to provide rapid analytics support during disasters (e.g., using satellite imagery for damage assessment). It has built local capacity in remote sensing and likely produced some publicly available disaster maps or tools.
* **SkAI-Pinas** (Sky Artificial Intelligence Program of the Philippines): This is a major AI initiative. It *“utilises remote sensing applications and big data”* to build AI solutions[[5]](https://archive.opengovasia.com/2025/06/13/the-philippines-significant-investment-to-advance-ai-ecosystem/#:~:text=Another%20major%20initiative%20is%20the,remote%20sensing%20applications%20and). SkAI-Pinas is essentially a multi-year program to harness AI for EO and other domains, and it includes sub-projects like those below.
* **ASTI Automated Labeling Machine (ALaM)**: (Instructor info – part of SkAI-Pinas) A project to automate and crowdsource the labeling of EO images to create training datasets. If participants ask, mention that creating labeled data is a big challenge and ASTI-ALaM tries to address that.
* **DIMER** (Democratized Intelligent Model Exchange Repository): Describe DIMER as an **AI model hub** developed by DOST-ASTI, aimed at making pre-trained AI models accessible. According to ASTI, DIMER *“lowers barriers to AI by enabling end-users to reuse optimized AI models”*[[6]](https://asti.dost.gov.ph/news-articles/asti-leads-ph-ai-revo-with-dimer-model-hub/#:~:text=...%20asti.dost.gov.ph%20%20DOST,users). In essence, it’s a repository where one might find ready-to-use models (e.g., for crop classification or flood detection) that can be downloaded or used via APIs. This is highly relevant for participants interested in applying AI without building everything from scratch.
* **AIPI** (AI Processing Interface): Explain that AIPI is an interface or platform that allows users to run AI models on data easily (likely tied to DIMER or the SkAI-Pinas infrastructure). It’s described as streamlining large-scale remote sensing tasks. In simpler terms, AIPI might let someone upload satellite imagery and apply an AI model (from DIMER) to it without deep coding – essentially an **AI inference service** for EO. This shows how the Philippines is developing tools to operationalize AI for everyone.

Encourage participants to see DOST-ASTI as a hub of innovation in EO and AI. If time permits, share any example outcome (e.g., a story of an AI model that maps something useful in the Philippines) to make it concrete.

* **Philippine Atmospheric, Geophysical and Astronomical Services Administration (PAGASA):** PAGASA is the national meteorological and hydrological agency. It provides **climate and weather data** – such as historical rainfall, typhoon tracks, forecasts – which can be very useful in EO projects (for example, combining Sentinel-1 flood maps with rainfall data, or using climate data as ancillary features in models). Mention any publicly accessible PAGASA data (like the Climate Data portal or APIs if available). The key point is that meteorological data can complement satellite observations in AI models (for instance, drought monitoring could combine rainfall deficit data with satellite vegetation indices).

After describing each, **highlight the synergy**: For robust EO applications, one often combines satellite imagery with local datasets. For example, an AI model for land cover mapping might be improved by including existing NAMRIA land cover maps for training or validation. Disaster mapping with AI could use PAGASA’s typhoon data to focus on affected areas. The Philippine agencies can provide context, ground truth, and platforms to distribute the results of AI models, ensuring solutions are actually adopted by local stakeholders.

*Instructor Tip:* Engage the class by asking if anyone has used data from these sources or is affiliated with these agencies. This can personalize the session and encourage knowledge-sharing among participants.

## **Activity: Introduction to the CopPhil Mirror Site and Digital Space Campus**

Finally, conclude Session 1 with a short interactive segment introducing two key resources established by the CopPhil program:

* **CopPhil Mirror Site:** Explain that this is a new **Philippines-based data center** that holds a **mirror of Copernicus data** focused on the Philippine region. Essentially, it will locally store Sentinel data (and possibly other Copernicus datasets), enabling faster and reliable access for users in the Philippines. Cite that *“this repository of data for the Philippines will be a copy of the Copernicus data made available on a fully scalable platform by 2025”*[[7]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=The%20major%20part%20of%20the,to%20be%20launched%20by%202025). In practical terms for participants: once operational, they could download Sentinel imagery from this local mirror much faster than from European servers. It will also reduce dependence on internet bandwidth because data is cached nationally. As the instructor, if you have access or screenshots, **demonstrate the web interface** of the mirror site (if it’s available in preview form) – show how one might search for a Sentinel-2 scene over the Philippines and initiate a download. If the site isn’t accessible yet, just describe its purpose and that PhilSA will host it with CloudFerro’s support, as part of the EU’s capacity building[[8]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=national%20interest,data%20management%20by%20Philippine%20stakeholders).
* **CopPhil Digital Space Campus:** Describe the Digital Space Campus as an online portal for **training materials and continued learning**. All the presentations, Jupyter notebooks, datasets, and guides from this 4-day training will be uploaded there for participants to revisit. The Campus likely also features additional learning resources and possibly forums or a community of practice. This is crucial for sustaining the impact of the training. As an instructor, you should show how to access the Digital Space Campus (e.g., through a PhilSA or project website, possibly requiring a login if it’s a learning management system). If possible, do a quick tour: show the section where training modules reside, how to download notebooks or slides, and any interactive features (like quizzes or discussion boards if present). Encourage participants to explore it after the session, emphasizing that it’s **their resource for self-paced study and for sharing knowledge with colleagues** who could not attend.

Wrap up the activity by reinforcing that these platforms – the Mirror Site for data and the Digital Space Campus for knowledge – are part of building a sustainable EO ecosystem in the Philippines. They reflect the investment being made (with EU support) to ensure participants can continue working with Copernicus data and advanced AI/ML tools *after* the course, without starting from scratch. This sets the stage for Day 2 and beyond, where they will dive deeper into practical AI/ML techniques using the data we discussed.

# Session 2: Core Concepts of AI/ML for Earth Observation

**Overview:** Session 2 covers fundamental Artificial Intelligence/Machine Learning concepts tailored to Earth Observation applications. It is mostly theoretical/conceptual, setting the groundwork for the hands-on work later. Instructors should clarify what AI/ML means in practice, walk through the typical workflow of an EO machine learning project, differentiate between types of ML (supervised vs unsupervised) with examples, introduce the basics of deep learning (neural networks), and stress the importance of data quality/quantity (“data-centric AI”) especially in the context of satellite imagery. The goal is to ensure everyone has a solid conceptual framework and vocabulary before proceeding to coding and model training in subsequent sessions.

## What is AI/ML? The AI/ML Workflow in EO

Start by demystifying **AI (Artificial Intelligence) and ML (Machine Learning)**. Define **Machine Learning** as a subset of AI where computer algorithms learn patterns from data **without being explicitly programmed with rules**. In Earth Observation, this means algorithms can learn to recognize features (like forests vs. water) from example satellite images rather than us hard-coding the spectral thresholds. You may mention that **AI** is a broader term that includes machine learning and other techniques (even rule-based systems), but in modern contexts AI often *implicitly* refers to machine learning approaches.

Outline the **typical workflow** of an AI/ML project in the Earth Observation domain. It helps to list these steps clearly for participants – consider writing them on a virtual whiteboard or as bullets in the presentation. For example:

1. **Problem Definition:** Identify the EO problem and objectives. (e.g., *“We want to classify land cover in Palawan”* or *“detect flooded areas after a typhoon”*). Being clear on the question helps design the solution.
2. **Data Acquisition:** Gather relevant data. In EO this often means satellite images (Sentinel, Landsat, etc.), but also includes ground truth labels or ancillary data. Explain that data can come from open sources (like Copernicus, NASA) or local agencies and that acquiring enough quality data is a major effort.
3. **Data Pre-processing:** Clean and prepare the data for analysis. For satellite imagery, this includes steps like atmospheric correction (if not using Level-2 data), cloud masking, radiometric corrections, geometric alignment, etc. Emphasize that *“garbage in, garbage out”* applies – well-prepared input data is crucial.
4. **Feature Engineering:** Especially for classical ML (not as much for deep learning with images), this means deriving informative features from raw data. In EO, features could be spectral indices (e.g., NDVI, NDWI), texture measures, topographic features (slope, elevation from DEM), etc. If doing deep learning on images, feature engineering is more automated (the CNN learns features), but for simpler ML like Random Forests, the practitioner chooses features.
5. **Model Selection and Training:** Choose an appropriate model (classification vs regression, etc.) and train it on the labeled data. Training involves feeding data to the algorithm to let it adjust its parameters. Mention that we’ll cover examples like Random Forest (Day 2) and CNN (Day 3) specifically for EO tasks.
6. **Validation and Evaluation:** After training, evaluate the model’s performance on independent test data. For EO, common validation includes computing confusion matrices, accuracy, precision/recall, etc., often using a separate set of ground truth points or areas. Stress *rigorous validation* – using proper held-out data or cross-validation to ensure the model generalizes well and isn’t overfitting.
7. **Deployment and Operationalization:** Finally, if the model is satisfactory, deploy it for use. In EO context, deployment could mean generating a full map (e.g., land cover map for entire region), setting up a pipeline to process new satellite images as they come (for near-real-time applications like deforestation alerts), or integrating the model into a decision support system. Also mention considerations like model retraining over time (especially if new data distributions come in, e.g., new sensor or landscape changes) and maintenance.

These steps form an iterative cycle – often you loop back, for instance, if validation is poor, you might collect more data or try a different model. Encourage questions here, as understanding the workflow is foundational.

*Instructor Note:* To keep this engaging, you could take a concrete case (say, *mangrove extent mapping*) and briefly illustrate each step in that context. For example: Problem – map mangroves; Data – Sentinel-2 imagery + training polygons of mangroves vs others; Pre-process – cloud mask images; Features – maybe use NDVI and water index; Model – train a Random Forest; Validate – check accuracy against known mangrove areas; Deploy – produce mangrove map for entire coastline and share with agencies. This makes the workflow less abstract.

## Types of Machine Learning: Supervised vs Unsupervised (with EO Examples)

Explain that there are different **paradigms of ML**. The two primary ones to cover are **Supervised Learning** and **Unsupervised Learning** (there are others like reinforcement learning, but those are beyond our scope and less common in EO at this level).

* **Supervised Learning:** The most common in EO. Define it as learning from **labeled data** – the algorithm is given examples with known outcomes (labels) and must learn to predict the label for new, unseen data. There are two branches:
* **Classification:** when labels are **categorical classes**. EO examples: land cover classification (labels like forest, water, urban, agriculture on pixels or image patches)[[9]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=Module%3A%20Types%20of%20ML%3A%20Supervised,with%20examples%20like%20image%20clustering), or cloud vs not-cloud detection in an image. Another example: identifying whether a given satellite image patch contains a certain crop type (classifying crops).
* **Regression:** when labels are **continuous values**. EO example: predicting a continuous variable like biomass or forest carbon from satellite data[[10]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=,with%20examples%20like%20image%20clustering), or estimating sea surface temperature from infrared imagery. Basically any prediction of a numeric value (e.g., soil moisture, air pollution level from satellites) is a regression task.

Emphasize that supervised learning needs **ground truth data** – e.g., in land cover mapping, we need training polygons or sample points of known land cover type to teach the model. The quality and representativeness of these labels directly influence model performance.

* **Unsupervised Learning:** Learning from **unlabeled data** by finding patterns or groupings inherent in the data. The typical example is **clustering**[[10]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=,with%20examples%20like%20image%20clustering). In EO, one might use clustering to automatically group pixels with similar spectral characteristics without knowing what they are – for instance, running *k*-means clustering on an image might segment it into clusters that (one hopes) correspond to different land cover types, but it’s up to the analyst to interpret cluster meaning (e.g., Cluster 1 = water, 2 = forest, etc.). Mention that unsupervised methods are useful for exploratory analysis – e.g., identifying that there seem to be X distinct spectral classes in an area – or for tasks like anomaly detection (finding pixels that look unusual). However, unsupervised results often need further refinement or labeling to be operationally useful.

Another unsupervised approach in EO is **dimensionality reduction** (like PCA) to find patterns in multi-band data, but keep things simple unless the audience is clearly advanced.

Make sure participants understand the distinction: supervised learning requires examples of “right answers” to learn from, unsupervised does not. You can use a quick analogy: *“Supervised learning is like a student learning with an answer key (they get feedback on right/wrong answers), whereas unsupervised is like a student figuring out patterns without any guidance.”*

EO Example to illustrate both: Suppose we have satellite images of a region and we want to identify different land cover. A **supervised** approach would use a training dataset of points/polygons labeled as forest, urban, etc., and train a classifier like Random Forest or a neural network to predict those classes for every pixel. An **unsupervised** approach might perform clustering on the image bands to group pixels into, say, 5 clusters based on reflectance similarity; we then might interpret those clusters as likely forest, water, etc., but without ground truth we don't know for sure – we might have to manually assign meaning or combine with some reference data to label the clusters.

It’s also worth noting that in practice, supervised methods generally yield more accurate and controlled results for classification tasks *if* good training data is available, which is why our focus in this workshop is on supervised learning (and eventually deep learning, which is essentially supervised for our tasks). Unsupervised methods like clustering can still be useful for quick, initial insights or for data compression.

## Introduction to Deep Learning: Neural Networks Basics

Now transition to **Deep Learning** as a subset of ML that has driven recent advances, especially in image analysis. Make sure to clarify that deep learning is essentially about **neural networks with many layers** (“deep” refers to multiple layers).

**Neural Network fundamentals:** Explain the building block – an **artificial neuron**. It’s a simple mathematical function: it takes inputs (say, pixel values or features), multiplies each by a weight, sums them up, adds a bias, and then applies an **activation function** (a non-linear function) to produce an output. A single neuron is like a logistic regression unit. These neurons are organized into layers: - **Input layer:** where data comes in (for an image, this could be pixel values for each band). - **Hidden layers:** one or more layers of neurons that progressively extract higher-level features. Each neuron in a hidden layer takes output from previous layer neurons as input, applies weights and activation. Emphasize that “deep” learning typically implies many hidden layers (dozens in state-of-the-art CNNs, but even a few layers counts as a neural network). - **Output layer:** produces the final predictions (e.g., probabilities of classes or a numeric value).

Mention that each connection has a weight that the network needs to learn. **Learning** (training) a neural network means finding the set of weights (and biases) that make the network’s predictions as accurate as possible.

Introduce key concepts in training neural networks: - **Loss function:** a metric of error that the network tries to minimize. For instance, mention *cross-entropy loss* for classification (the network compares its predicted class probabilities to the true class and incurs a penalty when it’s wrong) and *mean squared error* for regression (difference between predicted and actual values squared). It’s how we quantify “how bad” a particular set of predictions is. - **Optimizer:** the algorithm that adjusts the network’s weights to reduce the loss. The most common example is **Stochastic Gradient Descent (SGD)** – explain that it computes gradients of the loss with respect to each weight (using backpropagation) and nudges weights in the direction that lowers the loss. Also mention popular variants like **Adam** optimizer, which generally converges faster by adaptively tuning the learning rate. Participants don’t need to know the math details, but understanding that there is an automated process to tweak millions of parameters in a deep network is important. - **Epochs, Training process:** Typically we train in iterations (epochs) over the dataset, each time making the network a bit better at the task. The network gradually “learns” to approximate the mapping from inputs to outputs.

While keeping it conceptual, connect to EO: In image classification tasks (like identifying a flooded pixel vs not flooded), a deep neural network could take raw pixel values and learn by itself which combinations of spectral bands indicate “flood”. **Convolutional Neural Networks (CNNs)** are especially relevant for images – you might briefly mention that CNNs are neural networks specialized for grid data like images, using convolutional layers to automatically extract spatial features (edges, textures, shapes). We will cover CNNs more in Day 3, so here just sow the seed: deep learning excels at image recognition and has begun outperforming traditional methods in many EO tasks by automatically learning complex feature representations.

As an instructor, gauge the audience’s familiarity: if many look new to neural nets, use an analogy (e.g., neurons in brain, though artificial ones are vastly simpler). If some know this already, keep it tight but ensure the less experienced follow the basic idea.

Crucially, reassure participants that they don’t need to derive backpropagation equations; frameworks like TensorFlow and PyTorch handle that. The aim is to know the terminology (layers, activation, loss, optimizer) and the intuition that we are basically fitting a very flexible function to data.

## Data-Centric AI in EO: The Importance of Data Quality, Quantity & Diversity

Conclude Session 2 with an emphasis on the **data-centric approach** to AI, a concept popularized by Andrew Ng and very pertinent to EO. The principle: whereas a lot of early ML progress focused on model algorithms (model-centric approach), **Data-Centric AI** advocates that improving your *data* often yields the biggest gains in performance. In other words, the “food” you feed to your AI matters more than fancy tweaks to the model architecture[[11]](https://valohai.com/blog/data-centric-ai/#:~:text=Data,means%20that%20what%20you).

In the context of Earth Observation: - **Data Quality:** Satellite data can be noisy or have errors. Cloudy or hazy images, sensor artifacts, misregistration between image bands, etc., can all degrade model performance. Ground truth labels might also be of varying quality (e.g., mislabeled points, or polygons that don’t exactly align with image features due to GPS error or timing differences). Emphasize cleaning data: e.g., remove or mask clouds (we’ll practice that), ensure images are well coregistered, and validate training labels carefully. High-quality annotated data (even if smaller in quantity) can often beat a huge dataset of sloppy labels. - **Data Quantity:** Generally, more training data is better. Deep learning in particular is data-hungry. However, in EO, getting labeled data is expensive – you need experts or ground surveys. This is why techniques like data augmentation (synthetically increasing dataset via rotations, adding noise, etc.) or transfer learning (using models pre-trained on large datasets) are used. If appropriate, mention that in EO we also leverage *proxy data* – e.g., using high-resolution imagery to label lower-res imagery or using simulation. - **Data Diversity:** Models trained on a narrow dataset might fail when deployed on broader scenarios. For EO, this means if all your training images are from summer in Luzon, the model might struggle with winter scenes or with Mindanao if landscapes differ. Encourage having training data that covers the range of conditions the model will face: different seasons, different geographic regions, different sensor conditions. Diversity also means capturing various examples of each class (e.g., for “urban” class, have samples from big cities, small towns, different roofing materials, etc., so the model isn’t biased to one type). - **Annotation and Label Strategies:** Discuss the importance of *good labeling practices*. For example, if doing land cover classification, define classes clearly (what is “forest” vs “shrub”?). Ensure consistency in how humans label the data. Some projects fail because labelers had different interpretations. A data-centric approach might involve iteratively reviewing and improving labels (perhaps even using model results to find mislabeled examples).

You can cite that *“the success of AI/ML in EO is profoundly dependent on the data itself”*[[12]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=Module%3A%20Data,models%20underperform%20due%20to%20data), echoing that even the best algorithms will underperform if the training data is flawed. Instructors should share any personal anecdotes if available: e.g., *“We tried project X with a fancy model, but when we doubled our training data and cleaned it, the accuracy jumped from 70% to 90%.”* This drives home the point.

Finally, connect Data-Centric AI to participant’s work: encourage them to invest time in building or curating good datasets. For instance, if they’re interested in a certain application (like coral reef mapping or rice crop detection), they might spend substantial effort collecting ground truth or synthesizing training data – and that’s normal and important. The takeaway: **better data beats a cleverer model** in many cases, especially in remote sensing where unique challenges (atmospheric effects, sensor differences, etc.) mean a model could easily latch onto artefacts if the data/labels aren’t carefully handled.

*Transition:* Now that they have the foundational concepts, the next sessions will get more hands-on. Session 3 will ensure everyone is up to speed with Python and geospatial libraries, and Session 4 will introduce Google Earth Engine for large-scale data processing. Encourage participants to ask questions or clarify concepts now – later parts will assume this understanding of ML terms and workflow.

# Session 3: Hands-on Python for Geospatial Data (Intermediate Level)

**Overview:** In this 2-hour hands-on session, participants will practice using Python for geospatial data analysis, focusing on vector and raster data handling which are fundamental skills for EO tasks. The session uses **Google Colaboratory (Colab)** as the platform, ensuring everyone can run Python in the cloud without local setup. The instructor’s role is to guide the participants through setting up the Colab environment, revisiting some basic Python syntax if needed, and then diving into geospatial libraries: **GeoPandas** for vector data and **Rasterio** (with Matplotlib) for raster data. By the end, participants should be comfortable reading a shapefile or GeoJSON, inspecting it, and plotting it, as well as reading a raster image (like a Sentinel-2 subset), doing simple operations (such as subsetting bands, basic stats, maybe a calculation), and visualizing it.

## Setting up Colab: Environment, Google Drive & Packages

Begin the session by introducing **Google Colab** – an online Jupyter notebook environment provided by Google. Ensure everyone has the link (colab.research.google.com) and is logged in to their Google account.

**1. Creating a Colab Notebook:** Demonstrate how to create a new Python 3 notebook in Colab. Point out the interface: where to write code vs text (Markdown) cells, how to run cells (Shift+Enter), etc. Given this may be new for some, take it slowly at first.

**2. Connecting Google Drive (if needed):** Often in this training, we might have data files (like shapefiles or raster TIFFs) provided via Google Drive or want to save outputs there. Show how to mount Google Drive in Colab:

from google.colab import drive  
drive.mount('/content/drive')

After running, an authentication link will appear – walk through the steps (click link, select account, copy code, paste). Once done, explain that '/content/drive/MyDrive' now links to their Drive. If the training materials (like data files) are shared via Drive, instruct them on how to access (e.g., maybe a shared folder they need to copy to their Drive or use a shared link).

**3. Installing necessary packages:** Colab comes with many libraries pre-installed, but specialized geo libraries like GeoPandas or Rasterio might not be there by default. Demonstrate installation via pip:

!pip install geopandas rasterio shapely pyproj

(the ! lets you run shell commands in Colab; explain briefly). It may take a minute to install. Mention that geopandas likely pulls in fiona, gdal and other dependencies. If any installation issues arise (sometimes GeoPandas needs a runtime restart after install), be ready to assist. **Instructor tip:** You might have a prepared notebook where you’ve pre-installed these to catch any gotchas.

**4. Verifying setup:** After installation, import the libraries to ensure they work:

import geopandas as gpd  
import rasterio  
import matplotlib.pyplot as plt

If these import without error, the environment is ready. Also highlight that Colab notebooks have access to CPU (and GPU/TPU if enabled – not needed now, but note for later deep learning sessions, we might enable GPU acceleration via Runtime > Change Runtime Type > GPU).

By now, each participant should have their Colab environment ready with required packages. This setup is a one-time overhead; later notebooks will assume this environment.

## Python Basics Quick Recap

Since participants might have varied Python backgrounds, do a **brief recap of Python fundamentals**. Emphasize this is not a full lesson, just touching on syntax they’ll see in the code: - **Data Types:** Show examples of basic types (int, float, string) and data structures like **lists** and **dictionaries**. For example:

x = 5 # int  
name = "Alice" # string  
nums = [1, 2, 3] # list  
info = {"city": "Manila", "pop": 1.78} # dict

We will mainly use lists and dictionaries when handling geospatial data (e.g., list of values, dict of parameters). - **Control Structures:** Give a quick example of a for-loop and if-statement, as they might use these in analysis:

for i in [1,2,3]:  
 if i % 2 == 0:  
 print(i, "is even")  
 else:  
 print(i, "is odd")

This prints which numbers are even/odd. Many geospatial operations might use loops or list comprehensions, though libraries like GeoPandas often let you avoid explicit loops. - **Functions:** Show a simple function definition and usage:

def add(a, b):  
 return a + b  
  
result = add(2, 3)  
print(result) # 5

Reinforce indentation rules in Python (colons and indent to define blocks). - **Libraries (import):** Already partly covered with geopandas etc., but ensure they know to refer to library documentation or help() if needed. Python’s extensive libraries are a big reason it’s powerful for EO.

Keep this recap interactive – you can ask the room if they’re comfortable with these basics and adjust speed accordingly. If some are already proficient, this part will be quick. If many are new, don’t dive too deep; instead plan to help them by providing ready-to-run code in the following sections.

## Hands-On: Loading and Exploring Vector Data with GeoPandas

Now the real geospatial part begins. Explain that **GeoPandas** is an extension of pandas that makes it easy to work with vector geospatial data (points, lines, polygons with attributes). It leverages shapely for geometry and fiona for file access. It essentially allows use of DataFrame-like operations on geospatial data.

**Step 1: Load a vector dataset.** Use a sample dataset relevant to the training. For example, a shapefile of **Philippine administrative boundaries** (maybe the provinces or regions), or an **AOI polygon** for an area of interest. If a file is provided (e.g., philippines\_adm1.shp), demonstrate reading it:

gdf = gpd.read\_file('/content/drive/MyDrive/data/philippines\_adm1.shp')

*(Adjust path to where the file actually is; if it’s in the shared drive folder, use that path.)* GeoPandas can read many formats: Shapefile, GeoJSON, KML, etc. If using a GeoJSON from a URL, you could show gpd.read\_file(URL) as well, but using a local file is fine.

**Step 2: Inspect the GeoDataFrame.** Treat it like a pandas DataFrame:

gdf.head()

This will output the first few rows, showing columns like name of province, perhaps population, etc., and a geometry column which holds the shapes. Discuss what they see: each row is a feature (e.g., a province), geometry might be polygon or multipolygon. Use gdf.crs to show the Coordinate Reference System of the data:

print("CRS:", gdf.crs)

If it prints something like EPSG:4326, explain that means lat/long (WGS84). If it’s EPSG:xxx, tell them what projection that is if known. Reinforce that understanding CRS is important when overlaying data (but if not diving deep, at least ensure they know what CRS their data is in).

**Step 3: Simple data manipulations.** Show selecting a subset or filtering:

# Example: filter to a specific region by name  
region\_name = "Northern Mindanao"  
subset = gdf[gdf['REGION'] == region\_name]  
print(f"{region\_name} has {len(subset)} provinces:")  
print(subset['PROVINCE'])

(This assumes columns like REGION/PROVINCE exist; adapt to actual dataset.) The idea is to remind them how to filter rows in a GeoDataFrame by attribute values, which works just like pandas.

Maybe demonstrate a geospatial operation: if we have an AOI polygon (say a bounding box or a specific province polygon), use GeoPandas to filter points within it or do an intersection. For example:

# If we had a point file of cities, how to spatially join or filter those within a province:  
# cities = gpd.read\_file('phil\_cities.shp')  
# cities\_in\_region = gpd.sjoin(cities, subset, op='within')

But that might be too much at once; gauge if participants are following.

**Step 4: Plot the data.** GeoPandas makes plotting easy:

gdf.plot(column='REGION', figsize=(6,6), legend=True)  
plt.title("Philippines Admin1 by Region")  
plt.show()

This will produce a quick map. The column='REGION' and legend=True will color provinces by region category with a legend. If using a single AOI polygon, just plotting it will outline the polygon. The goal is to demonstrate visualization of vector data in Python.

At this point, ensure each participant manages to load and plot at least one vector dataset. This skill will be needed for examining training data or AOIs in later exercises (e.g., visualizing where their ground truth points are on a map).

*Teaching note:* Common pitfalls – missing files or incorrect path, so help them navigate the Drive path. Also, shapefiles actually are multiple files (.shp, .shx, .dbf, etc.) – ensure they have all parts in the provided data. If a CRS warning appears (like GeoPandas warning about no CRSs or something), explain not to worry for now, or how to set a CRS if needed (gdf.set\_crs('EPSG:4326', inplace=True) if you know it).

By the end of this, participants should see a colored map of the vector data and feel a small victory that they made a map with Python!

## Hands-On: Loading, Exploring, and Visualizing Raster Data with Rasterio

Next, tackle **raster data**, which is central for EO since satellite images are rasters. We use **Rasterio** – a powerful library for reading/writing geospatial rasters. It interfaces with GDAL under the hood.

**Step 1: Open a raster file.** Use the provided **Sentinel-2 L2A image subset** (e.g., a GeoTIFF file containing a portion of a Sentinel-2 tile). The subset might be a multi-band TIFF or a single band – clarify what the file is (for example, it could be a stack of several bands, or a full image where we’ll read one band at a time to save memory). Provide the path and do:

import rasterio  
src = rasterio.open('/content/drive/MyDrive/data/sample\_S2.tif')  
print(src.profile)

The src.profile will print metadata: width, height (in pixels), number of bands, data type, CRS, etc. Discuss these: e.g., “width 5000, height 5000 means this image is 5000x5000 pixels; count=4 means 4 bands included (perhaps Blue, Green, Red, NIR), dtype=uint16 means 16-bit integers (common for Sentinel reflectance scaled by 10000), CRS = EPSG:32651 (UTM zone 51N, if our AOI is Philippines Palawan for example).” Adjust specifics to the actual data at hand.

**Step 2: Read raster data.** Show how to read the pixel values into a NumPy array:

band1 = src.read(1) # read first band  
print(band1.shape, band1.dtype)  
print(band1) # printing the array (or a summary like mean/min)

This will output the array’s shape (e.g., (5000, 5000)) and dtype. Mention that read(1) reads band index 1 (Rasterio is 1-indexed for bands). If multi-band, we could read all by src.read() which returns a 3D array (band, row, col). But be cautious with memory if image is large. Perhaps restrict to a smaller window or downsample for demonstration if needed.

**Step 3: Basic raster operations.** - **Statistics:** Calculate simple stats to illustrate array math:

import numpy as np  
print("Band1 min:", band1.min(), "max:", band1.max(), "mean:", band1.mean())

This gives an idea of reflectance values range (Sentinel-2 L2A values typically 0-10000 scale for 0.0-1.0 reflectance). - **Cropping:** If the task mentions cropping, demonstrate how to crop using bounds or window. E.g., if we have an AOI polygon from the earlier GeoPandas step, we can use rasterio.mask:

from rasterio.mask import mask  
# suppose subset (from above) is a GeoDataFrame with our AOI polygon  
geom = [subset.geometry.values[0]] # list of shapely geometry  
out\_image, out\_transform = mask(src, geom, crop=True)  
print(out\_image.shape) # should be (bands, new\_height, new\_width)

This will clip the raster to the polygon’s extent. It returns an array of the same band count but only covering the polygon area, plus a transform (coordinate info).

Ensure to mention that mask sets pixels outside the polygon to 0 (or nodata) and the crop=True option trims the array to the polygon bounds[[13]](https://rasterio.readthedocs.io/en/stable/topics/masking-by-shapefile.html#:~:text=Masking%20a%20raster%20using%20a,this%20example%2C%20the%20extent). If participants find this complex, at least explain conceptually: cropping a raster means limiting to a region of interest, which is useful to reduce data size and focus analysis.

* **Resampling:** This might be advanced, but you can briefly show how to downsample or upsample. For example, to downsample by 2:
* from rasterio.enums import Resampling  
  new\_width = src.width // 2  
  new\_height = src.height // 2  
  band1\_half = src.read(  
   1,  
   out\_shape=(1, new\_height, new\_width),  
   resampling=Resampling.bilinear  
  )  
  print("Resampled shape:", band1\_half.shape)
* This uses Rasterio’s built-in resampling to scale the data to half resolution[[14]](https://rasterio.readthedocs.io/en/stable/topics/resampling.html#:~:text=Resampling%20%E2%80%94%20rasterio%201,using%20the%20bilinear%20resampling%20method). Highlight why resampling might be needed – e.g., matching resolutions of different bands (Sentinel-2’s 20m bands to 10m) or creating quick previews.

If this is too much, you can just describe it instead of running code, depending on time and participants’ comfort.

* **Computation (optional):** If time and interest permit, demonstrate a simple raster calculation like NDVI (Normalized Difference Vegetation Index). Assuming band4 = red, band8 = NIR in the file:
* red = src.read(3) # if 3rd band is red in our TIFF  
  nir = src.read(4) # if 4th band is NIR  
  ndvi = (nir.astype(float) - red.astype(float)) / (nir + red)  
  print("NDVI range:", ndvi.min(), ndvi.max())
* This might yield values around -1 to 1 (if using surface reflectance). NDVI is a great example of an index that participants might use in their projects, and it reinforces array operations.

**Step 4: Visualize raster data.** Use matplotlib to display the raster: - Single band (grayscale):

plt.figure(figsize=(5,5))  
plt.imshow(band1, cmap='gray')  
plt.colorbar(label='Band1 reflectance')  
plt.title("Sentinel-2 Band1")

This should show the image (if extremely large, maybe use a smaller subset or decimate it). If the image looks dark or all black, it might be because values are high (0-10000). One might normalize or stretch for display. For simplicity, you can divide by 10000.0 to get 0-1 range before plotting:

plt.imshow(band1/10000, cmap='gray')

Or use vmin/vmax in imshow to clip extreme values for better contrast.

* Color composite (if multi-band): Create an RGB composite using, say, bands 4-3-2 (if those correspond to R, G, B):
* rgb = np.stack([src.read(4), src.read(3), src.read(2)], axis=2)  
  rgb = rgb.astype(float)  
  rgb /= 10000.0 # scale to 0-1  
  plt.figure(figsize=(6,6))  
  plt.imshow(rgb)  
  plt.title("True color composite")
* This will show a true-color image. You might need to adjust contrast (maybe multiply by some factor or gamma correction) because reflectance 0-0.2 might appear dark. But it’s a nice visualization if it works.

At this stage, participants see an actual satellite image plotted. This often excites them – encourage them to interpret what they see (e.g., “Those dark patches are water, the bright green are vegetation, etc.”).

**Troubleshooting notes for instructors:** Sometimes displaying large images can overwhelm Colab (interactive plot might be slow). If that happens, consider showing a smaller subset or downsampled version for plotting. Also, if memory is an issue, be mindful with reading entire big arrays; use .read(1, window=...) to read a small window if needed for demonstration.

Finally, reinforce why these skills matter: *All the advanced AI/ML stuff we’ll do is built on these basics.* If they can’t load data or do basic inspections, they can’t train models. By mastering GeoPandas and Rasterio basics, they are becoming self-sufficient in handling geospatial data in Python – which is crucial for debugging and extending beyond what’s in this course.

Encourage participants to practice outside of class with their own data if possible: for example, try loading a GeoJSON they find online, or read a different satellite image. The more familiar they are with these tools, the easier Day 2 and Day 3’s coding exercises (which might involve manipulating training data and results) will be.

# Session 4: Introduction to Google Earth Engine (GEE) for Data Access & Pre-processing

**Overview:** In Session 4, participants will learn to leverage **Google Earth Engine (GEE)** for accessing and preprocessing large geospatial datasets, notably Sentinel imagery. Earth Engine is a cloud-based platform with petabytes of satellite data readily available[[15]](https://earthengine.google.com/#:~:text=Ready), which can be processed using simple scripts without needing to download everything. This session is split between using the Earth Engine **JavaScript Code Editor** (for interactive exploration) and the **Python API** (in Colab) to integrate with AI workflows. Key concepts introduced include Earth Engine data types (Images, ImageCollections, Features, FeatureCollections), filtering data by date/space, common preprocessing like cloud masking and making composites, and finally exporting data from Earth Engine for use in external environments (like downloading a prepared image to use in a local ML model). By the end, participants should appreciate how GEE simplifies the data acquisition stage of the AI/ML workflow and know some best practices for bridging GEE with Python AI tools.

## GEE Core Concepts: Image, ImageCollection, Feature, FeatureCollection, Filters, Reducers

Begin with a brief introduction to **Google Earth Engine**: It’s a platform that *“combines a multi-petabyte catalog of satellite imagery and geospatial datasets with planetary-scale analysis capabilities”*[[1]](https://earthengine.google.com/#:~:text=Google%20Earth%20Engine%20combines%20a,for%20academic%20and%20research%20use). Stress that GEE has **hundreds of datasets** (Sentinel, Landsat, MODIS, climate data, etc.) readily accessible, and computations run on Google’s servers (so it can handle huge tasks that would be impossible on a personal laptop). It has a JavaScript and Python API – we’ll glimpse both.

Explain the core data structures in Earth Engine: - **ee.Image:** In Earth Engine, an Image is a single geospatial raster layer or a multi-band raster. For example, one Sentinel-2 image (with all its bands) is an ee.Image. It’s not the actual pixels downloaded to your computer, but a reference to data in the cloud that can be manipulated. You can think of an ee.Image as analogous to a Rasterio dataset or a numpy array, with geospatial metadata. - **ee.ImageCollection:** This is one of the most powerful concepts – an ImageCollection is a *collection (stack/time-series) of images*. For example, the entire Sentinel-2 archive for the world is an ImageCollection in GEE (ee.ImageCollection("COPERNICUS/S2\_SR") for Level-2A imagery). You can filter an ImageCollection by date range, by geographic bounds, by metadata (like cloud cover percentage). Once filtered, you can perform operations on all images in the collection (like map a function to each image, or reduce them to a single image by composite). Emphasize how this simplifies time-series or multi-image processing – no need for loops downloading each image. - **ee.Feature:** A Feature in EE is a spatial data feature, like a point, line, or polygon with attributes. It’s analogous to a GeoPandas GeoSeries row. For instance, a single city location with a name property can be an ee.Feature (geometry + properties). - **ee.FeatureCollection:** A collection of Features, e.g., a set of many points or many polygons. Similar to a GeoPandas GeoDataFrame. Earth Engine has many feature collections in its data catalog too (like administrative boundaries, census data, etc., although imagery is the main focus). - **Filters:** Earth Engine provides filter objects to refine collections. Common filters include ee.Filter.bounds(AOI) to get items that intersect an area of interest, ee.Filter.date(start, end) to filter by date, or ee.Filter.eq('KEY','VALUE') to filter metadata by value. In the high-level API, we often use convenience methods like .filterBounds(), .filterDate(), or .filterMetadata() which under the hood use these filters. - **Reducers:** Reducers in EE are operations that **aggregate or summarize data**. For example, ee.Reducer.mean() can take multiple values (like multiple images or multiple pixels) and compute the mean. Reducers are used in two main contexts: temporal compositing (reducing an image collection to one image via mean, median, etc.) and spatial reduction (reducing pixel values over a region to a statistic). We’ll see an example when creating a composite image (using a median reducer across time) and when perhaps computing stats over an AOI.

Illustrate these concepts with a quick example in the Code Editor (just conceptually, or actually show code):

// JavaScript in GEE Code Editor (for demonstration)  
var dataset = ee.ImageCollection("COPERNICUS/S2\_SR")  
 .filterBounds(myAOI)  
 .filterDate('2021-01-01', '2021-12-31');  
print("Number of images:", dataset.size());

This snippet treats the Sentinel-2 SR ImageCollection, filters to images intersecting myAOI (which could be an ee.Geometry or ee.Feature we define) and the year 2021, then prints the count. Explain that operations like .size() or .mean() on a collection are done server-side. The user doesn’t see all images listed (unless you print(dataset) which would show an object, not data itself).

It’s important to mention that Earth Engine uses **lazy evaluation**: nothing is actually downloaded or computed until you ask for a result (like using print or exporting data). The print statements in the Code Editor request some info from the server (like size), but in general EE builds a computational graph and only executes when needed. This is advanced, but at least caution them that **you cannot treat ee.Image as a numpy array directly**; operations are done with EE methods, not by iterating in pure Python.

Also, highlight that Earth Engine requires users to sign up (you need an account to use the Code Editor or the Python API). Ensure all participants have done so, otherwise show them the sign-up page briefly (earthengine.google.com). For the workshop, likely they arranged accounts.

## Hands-On: Searching and Accessing Sentinel-1 & Sentinel-2 in GEE

Now, in practice, show how to find and load specific Sentinel datasets in Earth Engine: - **Earth Engine Data Catalog Search:** In the Code Editor or on the GEE Datasets webpage, search for “Sentinel-2”. The official name for Sentinel-2 Surface Reflectance is **COPERNICUS/S2\_SR**. For Sentinel-1 GRD: **COPERNICUS/S1\_GRD**. Demonstrate finding those. The dataset page (if you open it) provides details like bands, image properties, and example usage. Encourage participants to utilize these pages for any dataset they want to use (they give sample code and info on band names).

* **Filter by location and date:** Write a small script (can do this in the Code Editor JS or Colab Python). For first exposure, the Code Editor might be easier due to instant map visualization:
* var aoi = ee.Geometry.Point(120.98, 14.50); // example: a point (Lon, Lat) near Manila  
  // or ee.Geometry.Rectangle(lon1, lat1, lon2, lat2) for bounding box  
  var s2 = ee.ImageCollection("COPERNICUS/S2\_SR")  
   .filterBounds(aoi)  
   .filterDate('2021-11-01', '2021-11-30')  
   .filter(ee.Filter.lt('CLOUDY\_PIXEL\_PERCENTAGE', 20));  
  print("Images found:", s2.size());  
  var firstImage = s2.first();  
  print("First image date:", firstImage.date());  
  Map.centerObject(aoi, 10);  
  Map.addLayer(firstImage, {bands:['B4','B3','B2'], min:0, max:3000}, "First S2 image");
* This code filters Sentinel-2 images for November 2021 over an AOI point (which would cover some tile that includes that point), further filters to images with less than 20% cloud cover (using metadata property CLOUDY\_PIXEL\_PERCENTAGE), then gets the first image from that filtered collection and displays it as a true color layer on the map. The min:0, max:3000 is a stretch for visualization (since SR values up to ~10000, we cut at 3000 for better contrast).

Walk through this code: explain each step (aoi as geometry, filterBounds ties collection to AOI, filterDate is obvious, the cloud filter shows how metadata filters work). The Map.addLayer only works in the JS Code Editor, not in Colab. If doing this in Colab Python, you’d use something like geemap to display, but that might complicate things; better to use Code Editor for the interactive map, which is very intuitive.

* **Sentinel-1 example:** Similarly, show how to access Sentinel-1:
* var s1 = ee.ImageCollection("COPERNICUS/S1\_GRD")  
   .filterBounds(aoi)  
   .filterDate('2021-11-01', '2021-11-30')  
   .filter(ee.Filter.eq('instrumentMode', 'IW'))  
   .filter(ee.Filter.eq('orbitProperties\_pass', 'DESCENDING'))  
   .filter(ee.Filter.eq('polarization', 'VV')); // single polarization for simplicity  
  print("S1 images:", s1.size());  
  Map.addLayer(s1.median(), {min:-20, max:0}, "S1 VV Nov2021");
* Explain these filters: Sentinel-1 has properties for pass direction and polarization. In this example, we filter to Interferometric Wide mode (the common mode), descending orbit (just as an example, descending passes maybe at night vs ascending in morning – not crucial detail but shows filtering by metadata), and VV polarization. We then take a median composite of all those images in November and display it (the pixel values are in decibels dB after some normalization – so typical range ~ -25 to 0 dB for surfaces, hence the visualization range).

The map now would have a grayscale radar image. Point out differences: water is dark (low backscatter), urban bright (strong backscatter), etc., complementing the optical image.

Encourage participants to try adjusting filters or looking at different dates. **Important:** If participants are coding along in the Code Editor, ensure everyone has their **Earth Engine account enabled**. If someone doesn’t, pair them up or have them watch, because enabling a new account can take a bit.

This part shows how quick it is to get imagery without download. No need to fetch 1 GB of data – just filtering and visualizing on the fly.

## Hands-On: Basic Pre-processing in GEE – Cloud Masking, Composites, and Clipping

Now demonstrate some common **pre-processing tasks** on the fly in Earth Engine, using Sentinel-2 as the example (since optical imagery needs cloud handling):

**Cloud Masking:** Sentinel-2 SR images come with a QA band or the SCL band that indicates cloud pixels. A simple approach is to use the QA60 band (bitmask for clouds and cirrus in Level-1C, but for Level-2A, better to use SCL). However, a quick method: Earth Engine’s data catalog often provides a masking example. You can either: - Use the QA60 band: it’s a 16-bit integer where certain bits mean clouds. For L2A, bits 10 and 11 of QA60 correspond to opaque and cirrus cloud flags. An example mask function:

function maskS2clouds(image) {  
 var qa = image.select('QA60');  
 // Bits 10 and 11 are clouds and cirrus respectively.  
 var cloudBitMask = 1 << 10;  
 var cirrusBitMask = 1 << 11;  
 // Both flags set to zero means pixel is clear  
 var mask = qa.bitwiseAnd(cloudBitMask).eq(0).and(  
 qa.bitwiseAnd(cirrusBitMask).eq(0));  
 return image.updateMask(mask);  
}  
var s2\_clean = ee.ImageCollection("COPERNICUS/S2\_SR")  
 .filterBounds(aoi)  
 .filterDate('2021-01-01', '2021-12-31')  
 .map(maskS2clouds);  
var composite = s2\_clean.median().clip(aoi);  
Map.addLayer(composite, {bands:['B4','B3','B2'], min:0, max:3000}, "Median Composite 2021");

This code defines a function to mask clouds in each image (by setting a mask where QA60 bits indicate clear)[[16]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=common%20pre,clipping%20imagery%20to%20an%20AOI). We .map() that function over the ImageCollection, effectively removing cloudy pixels from all images. Then we take the median reduction over the year to get one composite image with hopefully minimal cloud (since median will pick median reflectance at each pixel, which likely comes from a cloud-free date given clouds are masked out as null). We also .clip(aoi) to cut the image to our area of interest polygon (if aoi was a geometry or FeatureCollection defining a region). Finally, display the composite.

Explain each step’s logic to the participants. The result is a pretty cloud-free image for 2021 in that region. The composite can be used as input for classification in Day 2 (if that’s the plan).

* Alternatively, mention that for a quick composite one could simply do s2.filter(...).median() and hope for the best, but without masking that median might still have cloud influence (though median is somewhat robust to outliers, but thick clouds could skew it). Cloud masking is the proper way.

**Clipping:** Already covered with .clip(aoi) above. Emphasize that .clip() is often used to limit processed images to exactly the AOI boundary, which looks nicer and is computationally efficient if the AOI is small relative to full scenes. Without clip, the composite covers the extents of all images considered (which can be large rectangles).

**Note on Reducers:** We used .median(), which under the hood is an ee.Reducer.median applied across the collection. You can mention other reducers like .mean(), .min(), .max(), or more complex ones (like standard deviation, percentiles etc. via reduce() function with specified reducer). For example, median vs mean composite differences (median is better for avoiding extreme values like bright clouds or sensor noise, mean might blur things).

At this point, participants should see how little code in GEE can accomplish tasks (cloud masking 100 images and compositing them would be a lot of work locally!).

**Switch to Python API (optional or demonstration):** If time allows, show that you can do the same from Python in Colab using the Earth Engine Python API. This requires initializing the API (authentication). - In Colab, you’d do:

import ee  
ee.Authenticate() # they follow the link, copy code, etc.  
ee.Initialize()

This only needs to be done once per session. If using a service account or already saved credentials, it may skip. Given time constraints, you might skip coding this live, but at least mention the steps.

* Then the Python code is almost one-to-one:
* aoi = ee.Geometry.Point([120.98, 14.50])  
  def maskS2clouds(image):  
   qa = image.select('QA60')  
   cloudBitMask = 1 << 10  
   cirrusBitMask = 1 << 11  
   mask = qa.bitwiseAnd(cloudBitMask).eq(0).And(  
   qa.bitwiseAnd(cirrusBitMask).eq(0))  
   return image.updateMask(mask)  
  s2\_clean = (ee.ImageCollection("COPERNICUS/S2\_SR")  
   .filterBounds(aoi)  
   .filterDate('2021-01-01', '2021-12-31')  
   .map(maskS2clouds))  
  composite = s2\_clean.median().clip(aoi)
* To actually visualize or use composite in Python, one could use geemap (which integrates with folium) or use composite.getInfo() which is not feasible for large images (since it tries to fetch data).
* Instead, one would **export** or sample from it, which leads to the next module.

The key takeaway: Earth Engine code can be written in both JS and Python. For interactive exploration and quick map viz, the Code Editor is fantastic; for integration with Python ML workflows, you can use the Python API (often combined with packages like geemap or just to export data for local processing).

## Exporting Data from GEE for External AI/ML Workflows

Finally, cover how to get data *out* of Earth Engine, since complex AI models (especially deep learning) will likely be trained outside EE (e.g., in TensorFlow on Colab) because Earth Engine itself is not designed to train arbitrary PyTorch models, etc. Also, participants may want to save results or processed data for offline use.

Methods to discuss: - **Export to Google Drive:** Earth Engine allows exporting images or tables to the user’s Google Drive. For example, to export the composite we created:

Export.image.toDrive({  
 image: composite,  
 description: 'Palawan\_S2\_2021\_composite',  
 region: aoi, // could be a geometry or coordinates for polygon  
 scale: 10,  
 crs: 'EPSG:32651',  
 maxPixels: 1e13  
});

This will initiate an export task (visible in the Code Editor Tasks tab). You must hit “Run” on the task and it will take some time and then the file (GeoTIFF) will appear in your Drive. Explain each parameter: scale: 10 sets resolution (10 m here, appropriate for Sentinel-2). If region is a large polygon, ensure maxPixels is set high enough (default 1e8 might be too low for big exports, so we often set a large number or use tileScale for heavy computations). Once in Drive, they can download it or directly read into Colab from Drive.

* **Export to Cloud Storage:** If some have Google Cloud accounts and large data, mention you can export to Google Cloud Storage similarly. Probably not needed for this training, Drive is simpler.
* **Export table (FeatureCollection) to Drive:** For example, if they collected some training sample points in Earth Engine, they can export those as a shapefile or CSV. E.g., Export.table.toDrive(collection: mySamples, format: 'CSV', fileNamePrefix: 'training\_points'). This is useful if they did some labeling in EE.
* **Downloading directly via URL (smaller data):** You can also use getDownloadURL() on an image for quick small area exports. For example:
* url = composite.getDownloadURL({'scale':10, 'region':aoi})  
  print(url)
* This provides a URL to directly download the image. However, for large images, this is not recommended (the link might expire or be huge; the Drive export is more robust).
* **geemap for interactive export:** If already using geemap, there are helper functions like geemap.ee\_export\_image(composite, filename='image.tif', scale=10, region=aoi) that wraps the above process in Colab. Not mandatory to cover if short on time, but let advanced users know such tools exist.

Discuss Earth Engine limitations briefly: GEE is great for data prep, but it has quotas and not suited for heavy *training* of deep models. For instance, you cannot run a training loop of a PyTorch model on GEE servers. GEE can do some simple ML (it has built-in classifiers like CART, SVM, even a way to do TensorFlow inference if model is exported), but for our advanced uses, we typically use GEE to get data ready (like export patches, compute indices, etc.), then use Python with ML libraries outside GEE. This workflow – GEE for data and Python for modeling – is exactly what we want them to learn.

Finally, mention **best practices** when moving between GEE and external: - Be mindful of **projection and resolution** when exporting (choose appropriate CRS and scale to match what your model expects). - Be careful with **data size** – exporting a huge area at 10m resolution will create a massive file. Sometimes better to break into tiles or use manageable AOIs. - Use **Drive or Cloud** depending on size: Drive has a 5TB overall quota and maybe slower for huge files, Cloud Storage is better for very large. - Always verify exported data alignment and values (sometimes needs scaling or bit decoding). - Clean up after yourself (if they keep making exports, their Drive might fill up etc., so delete things when not needed).

**Activity (if time)**: Let participants attempt an export of a small region composite themselves. For example, have them define a polygon around their hometown, create a median S2 composite of last year, and export to Drive. They can later load that TIFF in QGIS or with Rasterio to check – a nice full-circle of cloud to local.

Wrap up Session 4 by highlighting that now they have the tools to get **almost any EO dataset** they need and pre-process it for AI. This solves the common bottleneck of data availability and cleaning. In Day 2, they’ll apply these data and skills to actually train a machine learning model (Random Forest classification), and in Day 3-4 move to deep learning and advanced topics.

Encourage them to experiment with Earth Engine for their areas of interest – it has datasets beyond Sentinel (Landsat, MODIS, etc.) and even a lot of climate and vector data. Mastering the few concepts we covered opens up a world of analysis at planetary scale.

[[1]](https://earthengine.google.com/#:~:text=Google%20Earth%20Engine%20combines%20a,for%20academic%20and%20research%20use) [[15]](https://earthengine.google.com/#:~:text=Ready) Google Earth Engine

<https://earthengine.google.com/>

[[2]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=focusing%20on%20the%20Sentinel,Engine%2C%20will%20be%20thoroughly%20explained) [[9]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=Module%3A%20Types%20of%20ML%3A%20Supervised,with%20examples%20like%20image%20clustering) [[10]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=,with%20examples%20like%20image%20clustering) [[12]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=Module%3A%20Data,models%20underperform%20due%20to%20data) [[16]](file://file-Ew3VDK9LGP7AkVwLk279uj#:~:text=common%20pre,clipping%20imagery%20to%20an%20AOI) CopPhil EO AI\_ML Training Agenda - Final - 040725.docx

<file://file-Ew3VDK9LGP7AkVwLk279uj>

[[3]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=stakeholders) [[4]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=Copernicus%20is%20the%20European%20Earth,applications%20based%20on%20satellite%20data) [[7]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=The%20major%20part%20of%20the,to%20be%20launched%20by%202025) [[8]](https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/#:~:text=national%20interest,data%20management%20by%20Philippine%20stakeholders) CloudFerro supports ESA project for the Philippines

<https://cloudferro.com/news/cloudferro-supports-esa-project-for-the-philippines/>

[[5]](https://archive.opengovasia.com/2025/06/13/the-philippines-significant-investment-to-advance-ai-ecosystem/#:~:text=Another%20major%20initiative%20is%20the,remote%20sensing%20applications%20and) The Philippines: Significant Investment to Advance AI Ecosystem

<https://archive.opengovasia.com/2025/06/13/the-philippines-significant-investment-to-advance-ai-ecosystem/>

[[6]](https://asti.dost.gov.ph/news-articles/asti-leads-ph-ai-revo-with-dimer-model-hub/#:~:text=...%20asti.dost.gov.ph%20%20DOST,users) DOST-ASTI leads Philippine AI Revolution with innovative DIMER ...

<https://asti.dost.gov.ph/news-articles/asti-leads-ph-ai-revo-with-dimer-model-hub/>

[[11]](https://valohai.com/blog/data-centric-ai/#:~:text=Data,means%20that%20what%20you) Data-Centric AI and How to Adopt This Approach - Valohai

<https://valohai.com/blog/data-centric-ai/>

[[13]](https://rasterio.readthedocs.io/en/stable/topics/masking-by-shapefile.html#:~:text=Masking%20a%20raster%20using%20a,this%20example%2C%20the%20extent) Masking a raster using a shapefile — rasterio 1.4.3 documentation

<https://rasterio.readthedocs.io/en/stable/topics/masking-by-shapefile.html>

[[14]](https://rasterio.readthedocs.io/en/stable/topics/resampling.html#:~:text=Resampling%20%E2%80%94%20rasterio%201,using%20the%20bilinear%20resampling%20method) Resampling — rasterio 1.4.3 documentation - Read the Docs

<https://rasterio.readthedocs.io/en/stable/topics/resampling.html>