Deep Learning Global Health Analytics
Software Installation and Configuration Guide

This guide covers software installation, file system structure, pre-processing pipelines, the main processing pipeline for unsupervised cluster analysis, and associated notebook configuration settings. It explains how to set up the required Python development environment on a Linux system and configure the project-specific software available on GitHub. The document also provides an overview of the project's file system structure, detailing the notebooks used for pre-processing geospatial data, satellite imagery, and (optionally) model fine-tuning. Although data is not stored in the repository due to its large size, the designated folders for storing specific datasets are clearly identified. All installation and configuration steps have been tested on macOS and Ubuntu.

1. Development Environment Configuration

The first step in the installation process is to create a conda virtual environment using Python 3.9 with the following command. The '$' represents the terminal command prompt:

```
~$ conda create -n py39-pt-test python=3.9
```

Activate the environment as follows:

```
~$ conda activate py39-pt
```

Once activated, the command line will reflect the activated environment: `(py39-pt) ~$`

Next, install the following Python packages using conda:

```
(py39-pt) ~$ conda install pytorch torchvision -c pytorch
(py39-pt) ~$ conda install numpy pandas pyreadstat
(py39-pt) ~$ conda install geopandas pyproj rasterio shapely gdal
(py39-pt) ~$ conda install tensorboard tqdm
(py39-pt) ~$ conda install matplotlib seaborn bokeh pillow
(py39-pt) ~$ conda install scikit-learn umap-learn
(py39-pt) ~$ conda install ipykernel jupyter
(py39-pt) ~$ conda install glob
(py39-pt) ~$ conda install selenium
```

Some packages are not available through `Conda` and must be installed using `pip` as follows:

```
(py39-pt) ~$ pip install torchinfo torchmetrics torchgeo scikit-gstat
```

To register the current Python environment as a Jupyter kernel, so it can be selected within the Jupyter interface, execute the following command (this is a one-time step):

```
(py39-pt) ~$ python -m ipykernel install --name=py39-pt --display-name "Python (py39-pt)"
```

Before launching the Jupyter notebook server, change directory to the top level project directory indicated below:

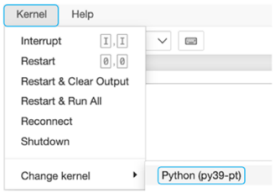
```
(py39-pt) ~/Deep-Learning-Global-Health-Analytics$
```

Next, issue the following command to launch the Jupyter Notebook interface in your default web browser. This can be executed from the command terminal from within the conda virtual environment or directly from the command terminal from outside the virtual environment.

```
(py39-pt) ~/Deep-Learning-Global-Health-Analytics$ jupyter notebook &
```

Since the kernel was registered to the virtual environment in the previous command, it can be selected from within the notebook.

After issuing the above command, you can launch any of the Jupyter notebooks in the project. However, before executing the code cells, make sure to go to the **"Kernel"** pulldown menu and select **"Change Kernel"** to confirm or set the kernel associated with the Conda virtual environment created earlier.



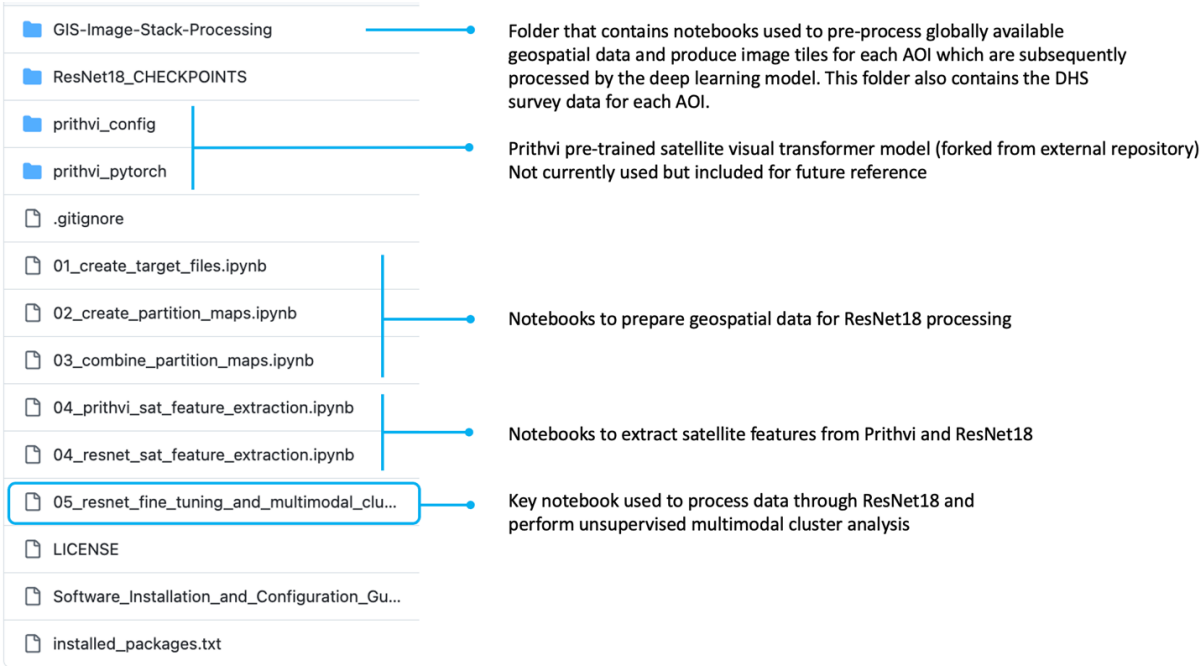
Note that the interaction with Jupyter notebooks may differ slightly depending on whether you're using **JupyterLab** or **Jupyter Notebook** (both interfaces for running Jupyter kernels). JupyterLab offers a more modern and flexible interface, but the code should execute the same in either environment.

2. Project Software

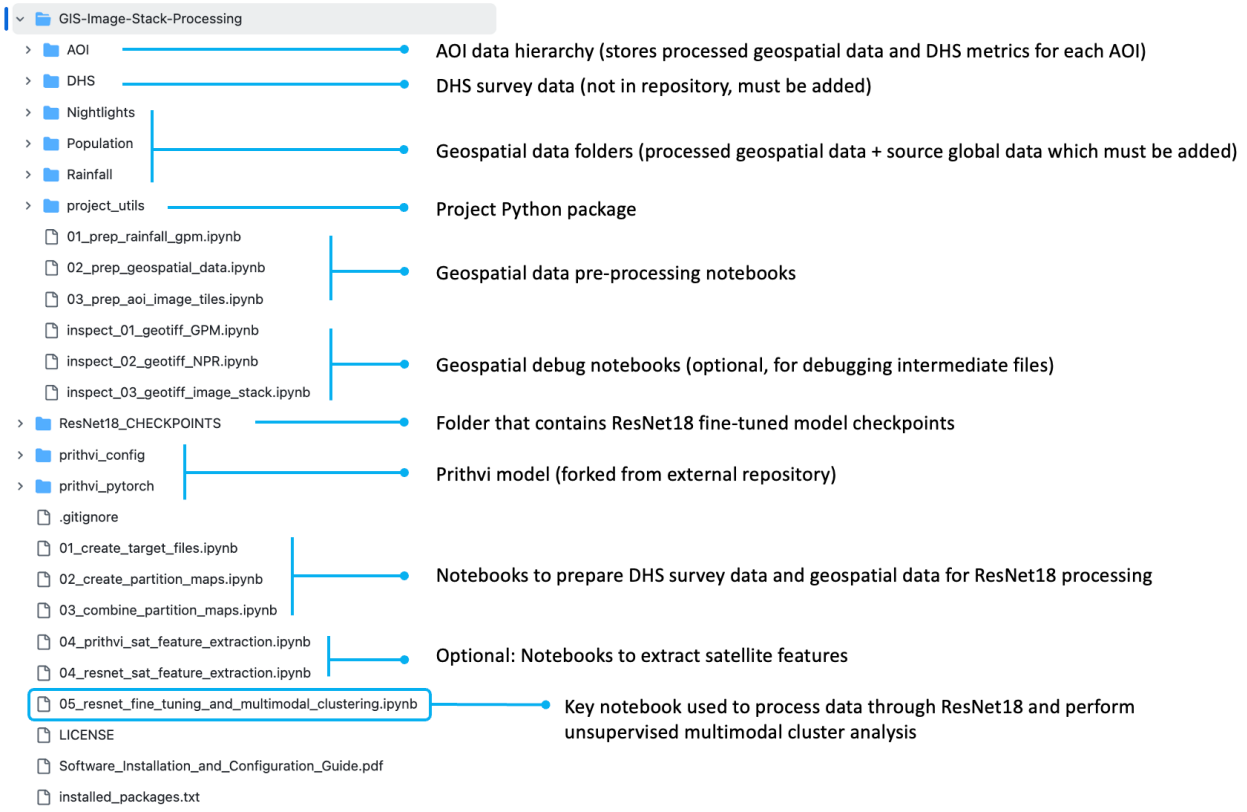
The software for this project is located at the following GitHub repository:

https://github.com/kromydas/Deep-Learning-Global-Health-Analytics/

The top-level folder hierarchy is shown below which identifies key folders and notebooks used to process geospatial data and satellite imagery.



The expanded view below shows the content of `GIS-Image-Stack-Processing`, which contains several notebooks used to pre-process DHS survey data and geospatial data. This folder also contains a project Python package `project_utils` with lower-level functions that support most of the notebooks in the repository.



The project Python package `project_utils`, contains many lower-level functions. The advantage of the Python package is that it allows these functions to be defined outside of the notebooks, reducing code clutter within the notebooks. The `project_utils` package consists of the following modules:

```
project_utils/
        aoi_configurations.py
        cluster_stats.py
        gist_utils.py
        plot_utils.py
        resnet_utils.py
        satellite_dataset_utils.py
```

There is also code related to the deep learning model Prithvi-100M, a pre-trained satellite visual transformer. This code was forked from the following repository:

https://github.com/isaaccorley/prithvi-pytorch

This code supports the use of the pre-trained model for processing satellite imagery and feature extraction.

2.1 GIS-Image-Stack-Processing

There are two main sets of notebooks associated with this project. The first set is located in `GIS-Image-Stack-Processing` and are used to pre-process globally available geospatial data and produce image tiles for each AOI which are subsequently processed by the deep learning model. This set of notebooks therefore serve the purpose of preparing the geospatial data, but only need to be executed "once" as a set, although each notebook is executed multiple times , as described below to generate the image tiles for each AOI.

Notebook	Notes
01_prep_rainfall_gpm.ipynb	**Pre-process global GPM rainfall data**. Converts multi-year monthly averages to a single global daily average for each AOI. This notebook should be executed once for each AOI. The output from this notebook is a daily average rainfall GeoTiff file for each AOI as shown in the example below: ./GIS-Image-Stack-Processing /Rainfall /GMP_2001-2022/ PK/AOI_Crop_Daily/ GPM_2001-2022.01.V07B_PK_avg.tif
02_prep_geospatial_data.ipynb	**Pre-process each geospatial data type for each AOI**. Creates AOI spatially aligned image stacks at a consistent and specified resolution. This notebook should be executed once for each data type for each AOI. For example, for the PK AOI specified, it should be executed once for Nightlights, once for Population and once for Rainfall. The output files from each execution represent the three channels in the AOI image stack (i.e., three large GeoTiff files covering the entire AOI).
03_prep_aoi_image_tiles.ipynb	**Create spatially aligned image tiles**: This notebook uses the AOI image stack to create smaller image tiles centered at the DHS survey locations. The AOI image stack from the previous step is copied to a new location on the file system intentionally for use with this notebook. The output from this notebook are 224x224 image tiles (GeoTiff files) located within each AOI. ./AOI/PK/Image_Tiles/ Nightlights/ PK_1_C-1_Nightlights_2022_400m.tif : Population/ PK_1_C-1_Population_2022_400m.tif : Rainfall/ PK_1_C-1_Rainfall_2001 2022_400m.tif :

2.2 Geospatial Data Preparation

The next set of notebooks prepare the geospatial data for ResNet18 processing. This process involves loading DHS survey data, computing cluster-level statistics for each AOI, and storing that data in JSON files for downstream processing. Plots are also generated to visualize the spatial distribution of the metrics, and variograms are produced as well. The target files for each AOI (e.g., `./AOI/PK/Targets/targets.json`) are then used as input (along with a specified criterion) to create virtual partition maps for training and validation, supporting model fine-tuning.

Notebook	Notes

`01_create_target_files.ipynb`	**Create target files**: This notebook loads and processes the DHS recode files for the specified AOI, computing cluster-level metrics. These metrics, along with the cluster IDs and their (lat, lon) coordinates, are stored in a `targets.json` file. Additionally, survey metrics are plotted on a geographical map to visualize their spatial distribution. Variograms for each metric are also generated. The resulting plots are saved to disk in the following directories, which are automatically created if they do not already exist.
	
	`Plots_Geospatial/`
	`Plots_Variograms/`
	
	Since the notebook serves multiple purposes the creation of target files and geospatial maps are both configurable.

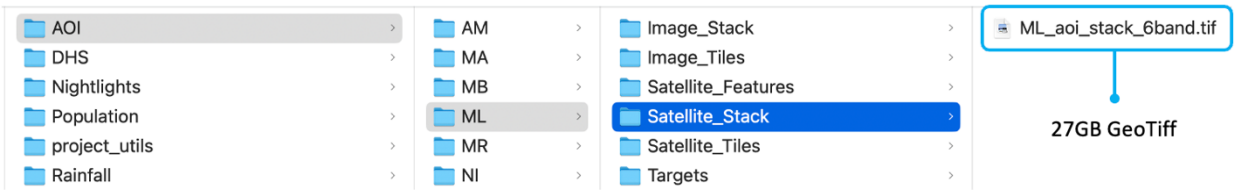
`02_create_partition_maps.ipynb`	**Create virtual partition maps**: This notebook uses DHS cluster data to virtually partition the clusters into training and validation sets. The current partitioning criterion applies a longitude threshold to minimize spatial autocorrelation between the training and validation datasets, though this can be adjusted to use other methods. The notebook generates three JSON files in the following location:
	
	`./GIS-Image-Stack Processing/`
	`AOI/`
	`Partitions/`
	`{country_code}/`
	`all.json`
	`train.json`
	`valid.json`
	
	Each file contains a JSON object where the two-letter country code is the key, and the value is an array of integers representing the cluster IDs for that partition. This structure allows for flexible, virtual data partitioning, making it easy to modify the partitioning logic as needed.

Notebook	Notes

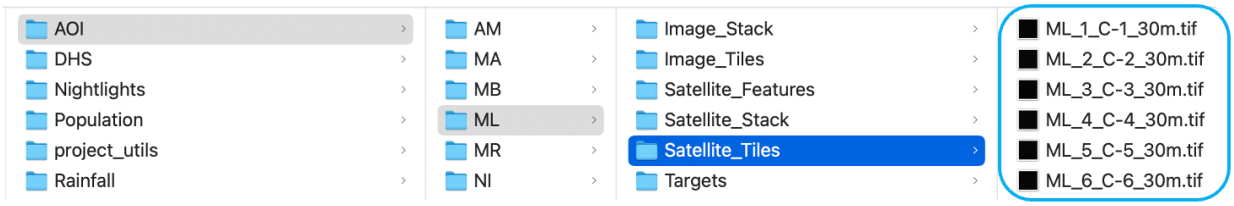
`03_combine_partition_maps.ipynb`	**Combine partition maps**: This notebook combines the AOI-specific partition maps into a master set based on the provided list of AOIs. The master partition maps are located directly within the Partitions folder:
	
	`./GIS-Image-Stack-Processing`
	`/AOI/`
	`Partitions/`
	`all.json`
	`train.json`
	`valid.json`
	
	These master partition files are used to create training and validation datasets for fine-tuning deep learning models. The AOI-specific partition maps generated by the previous notebook serve as a pre-processing step to enable the aggregation of the AOI-specific maps into a master set. The master `all.json` file is not used for training but represents the superset of all cluster IDs for a given AOI and serves as a reference.

2.3 Satellite Feature Extraction [Optional]

Satellite imagery is currently available for Mali and Pakistan. The process and software used to acquire the satellite imagery is beyond the scope of this document, however, the process produces a single large 6-channel GeoTiff file that covers the DHS locations for both AOIs. These GeoTiff files are stored in the `'Satellite_Stack'` folder corresponding to each AOI and serve as input for generating satellite tiles at individual DHS locations. The files are quite large as shown for Mali below.



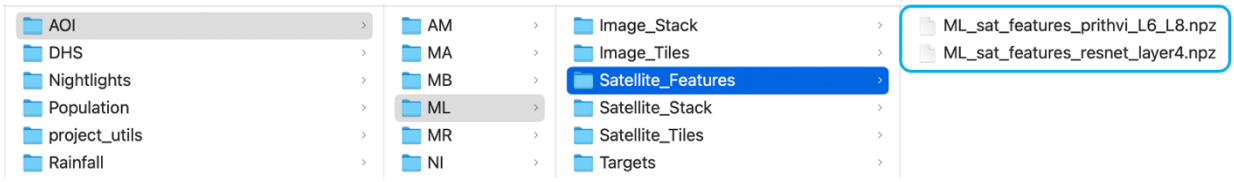
An additional processing pipeline is then used to crop satellite tiles at each DHS location from this image stack and store them in the `'Satellite_Tiles'` folder as shown below:



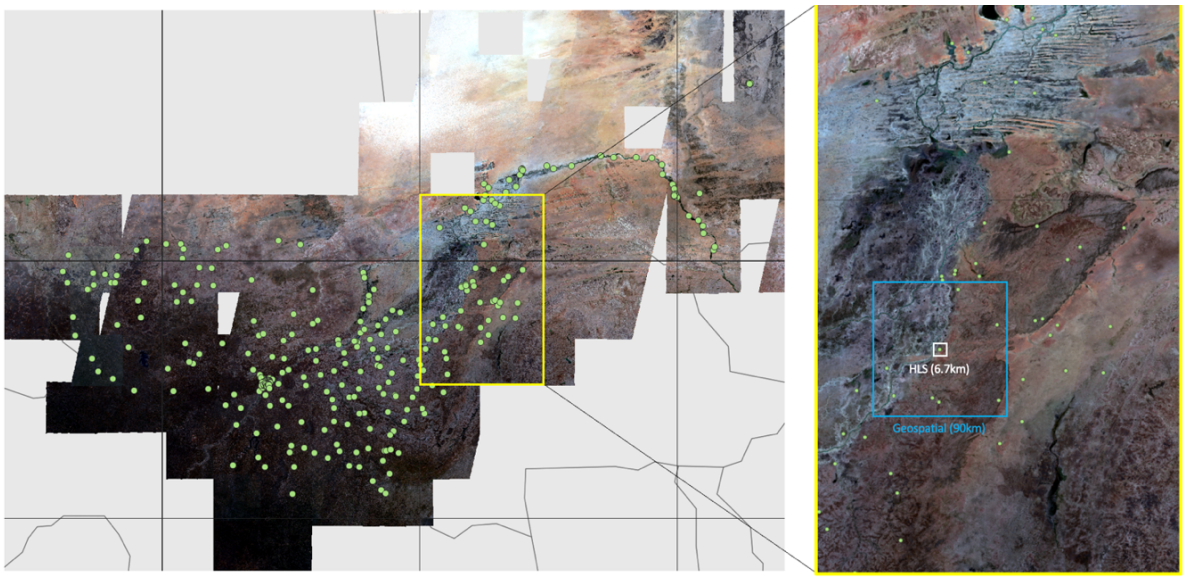
Two optional notebooks can then be used to process these tiles through pre-trained models (Prithvi or ResNet18) to extract features and save them to disk. The Prithvi model processes all six channels while the ResNet18 model only loads and process the RGB channels.

`04_prithvi_sat_feature_extraction.ipynb`
`04_resent_sat_feature_extraction.ipynb`

When these notebooks are executed, they will produce the following feature files as shown below. Each notebook allows the user to specify the AOI and the extraction layer(s) within the model. These are the only user specified configurations required. The resulting features are then saved to disk as shown below which also indicate the extraction points specified (e.g., `'L6_L8'` and `'layer4'`).



In the example below for Mali, the image on the left shows the HLS satellite coverage as a single composite GeoTIFF satellite image stack, overlaid with DHS cluster locations. The image on the right provides an expanded view of the region highlighted by the yellow rectangle, illustrating the level of detail in the satellite imagery. At each DHS location, satellite image tiles are generated for each cluster (30m resolution x 224 = 6.7km tiles). For comparison, the approximate sizes of the HLS (white) and geospatial (blue) image tiles are also shown for a single DHS cluster.



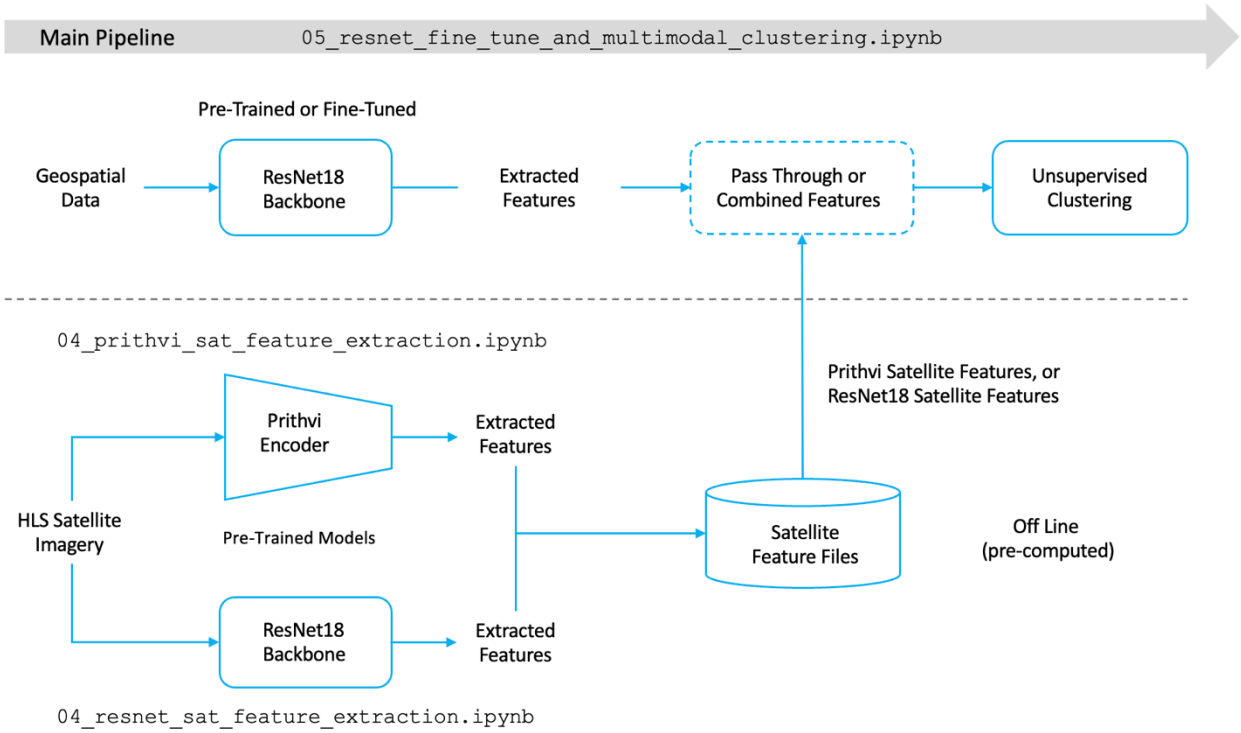
2.4 ResNet Fine-Tuning and Multimodal Cluster Analysis

The primary analysis notebook leverages the ResNet18 deep learning model to process geospatial data and perform unsupervised cluster analysis on the extracted features. This section describes the general workflow and configuration options for this notebook.

```
05_resnet_fine_tuning_and_multimodal_clustering.ipynb
```

The notebook, can be configured to use a pre-trained ResNet18 model, a checkpoint from a fine-tuned model, or to perform fine-tuning using target values associated with the input geospatial data at each DHS cluster location. In all cases, either the pre-trained model or a fine-tined version is then used to extract geospatial features from the model backbone to perform unsupervised cluster analysis. The notebook also offers the capability to load satellite features derived from satellite tiles at each DHS location if such data is available. The figure below summarizes the overall workflow, illustrating how pre-

computed satellite features can be loaded from disk and processed through the main pipeline. This supports unsupervised clustering analysis either by passing through geospatial or satellite features independently or by combining both feature sets for cluster analysis.



The notebook supports three models: the user can choose to use a pre-trained ResNet18 model, load a checkpoint from a fine-tuned model, or perform fine-tuning to create a new checkpoint. Additionally, a configuration setting is required to specify the type of features to use (geospatial, satellite, or both). With this introduction, we can now review the notebook configuration settings.

AOI Configuration

Before discussing the details of the above configuration settings, it's important to define the AOI at the top of the notebook using its two-letter country code. Regardless of the model type, an AOI must be specified for unsupervised cluster analysis. For example, in the configuration below, Pakistan is specified as the AOI. If model fine-tuning is performed, data from all AOIs is used for training, however, only the selected AOI is used for unsupervised cluster analysis with the specified model type.

```
country_code  = 'PK'      # Set the country code to one of the available AOIs in the list below

Available AOIs: AM (Armenia)
                MA (Morocco)
                MB (Moldova)
                ML (Mali)
                MR (Mauritania)
                NI (Niger)
                PK (Pakistan)
                SN (Senegal)
                TD (Chad)
```

```
#-----------------------------------------------
# REQUIRED CONFIGURATIONS HERE
#-----------------------------------------------
country_code  = 'PK'      # Set the country code
#-----------------------------------------------
```

Feature Type Selection

The next section in the notebook allows the user to specify which types of features should be used to perform unsupervised cluster analysis. The nominal configuration is to use geospatial features which are derived from the geospatial data that is available for each AOI. In this mode, the notebook extracts features from the model and performs several processing steps to cluster the data and produce various metrics and plots. This is all performed using geospatial data and does not require any external files other than the pre-computed geospatial image tiles and DHS survey data for each AOI.

If satellite data is available for a given AOI (specifically satellite tiles), then satellite features can be pre-computed using separate feature extraction notebooks using either a pre-trained ResNet18 or pre—trained Prithvi-100M model. Those notebooks process satellite tiles at each DHS location and save the extracted features to disk which can then be loaded in this notebook.

The following section in the notebook is used to configure which type of features should be used (geospatial, satellite, or both). The comments below describe the options.

```
Set the feature type (GEO, SAT, or ALL):

    features = FEATURES.SAT

If satellite features are available set the model type they were extracted from (RESNET or PRITHVI):

    sat_type = SAT_MODEL_TYPE.PRITHVI
```

```python
# The following enum types define the options for which features will be processed. Geospatial
# features will always be available,however, satellite feature availability is AOI dependent.

from enum import Enum
class FEATURES(Enum):
    GEO = "Geo"
    SAT = "Sat"
    ALL = "Geo-Sat"

class SAT_MODEL_TYPE(Enum):
    PRITHVI = "Prithvi"
    RESNET  = "ResNet"

#-------------------------------------------------------------------------------------------
# *** Specify which feature types to use. Geospatial, Satellite, or Both.
#
# Note: The ResNet18 model can only be fine-tuned in this notebook when using geospatial
#       data. However, geospatial features can still be combined with satellite features
#       as described below. Whether a pre-trained ResNet18 model is used to process
#       geospatial data or a fine-tuned version (or checkpoint), both use cases support
#       satellite feature integration when `features = FEATURES.ALL` is selected.
#
#       When `features = FEATURES.SAT`, only satellite features will be used for
#       unsupervised cluster analysis, even though the geospatial features are still
#       processed through the ResNet18 model. In other words, the feature configuration
#       acts as a global switch for cluster analysis. However, the code for processing
#       geospatial features through ResNet18 is not currently bypassed when
#       `features = FEATURES.SAT`. While this introduces some inefficiency, it simplifies
#       the logic by avoiding the need to bypass geospatial functionality below.
#-------------------------------------------------------------------------------------------

features = FEATURES.ALL              # SET THIS VALUE TO: GEO, SAT, or ALL
sat_type = SAT_MODEL_TYPE.PRITHVI    # SET THIS VALUE TO: RESNET or PRITHVI

#-------------------------------------------------------------------------------------------
#
# Valid combinations are:
#
# features = FEATURES.GEO
# sat_type = SAT_MODEL_TYPE.RESNET   # If features = FEATURES.GEO, sat_type is ignored
#
# features = FEATURES.SAT
# sat_type = SAT_MODEL_TYPE.RESNET
#
# features = FEATURES.SAT
# sat_type = SAT_MODEL_TYPE.PRITHVI
#
# features = FEATURES.ALL            # GEO + SAT
# sat_type = SAT_MODEL_TYPE.RESNET
#
# features = FEATURES.ALL            # GEO + SAT
# sat_type = SAT_MODEL_TYPE.PRITHVI

#-------------------------------------------------------------------------------------------
# *** Specify satellite feature file names
#-------------------------------------------------------------------------------------------
prithvi_sat_feature_file = f'{country_code}_sat_features_prithvi_L6_L8.npz'
resnet_sat_feature_file  = f'{country_code}_sat_features_resnet_layer4.npz'
#-------------------------------------------------------------------------------------------
```

Specification of feature Types to use

External satellite feature files

ResNet Geospatial Feature Extraction Layer Specification

The section below outlines the first half of the ResNet18 model configuration settings in the notebook. Only the final section requires editing to specify the feature extraction layer from the ResNet18 backbone, which is relevant only if the feature configuration includes geospatial data (i.e., `FEATURES.GEO` or `FEATURES.ALL`).


```python
# Enumerated list of DHS target values
class TargetType(Enum):
    FRACTION_DPT3_VACCINATED  = "fraction_dpt3_vaccinated"
    FRACTION_WITH_ELECTRICITY = "fraction_with_electricity"
    FRACTION_WITH_FRESH_WATER = "fraction_with_fresh_water"
    MEAN_WEALTH_INDEX         = "mean_wealth_index"
    FRACTION_WITH_RADIO       = "fraction_with_radio"
    FRACTION_WITH_TV          = "fraction_with_tv"


target_name_mapping = {
    TargetType.FRACTION_DPT3_VACCINATED:  "Fraction DPT3 Vaccinated",
    TargetType.FRACTION_WITH_ELECTRICITY: "Fraction with Electricity",
    TargetType.FRACTION_WITH_FRESH_WATER: "Fraction with Fresh Water",
    TargetType.MEAN_WEALTH_INDEX:         "Mean Wealth Index",
    TargetType.FRACTION_WITH_RADIO:       "Fraction with Radio",
    TargetType.FRACTION_WITH_TV:          "Fraction with TV",
}

class ModelMode(Enum):
    PRE_TRAINED = "Pre_Trained"
    FINE_TUNE   = "Fine_Tune"
    CHECKPOINT  = "Checkpoint"

# Dataset configuration parameters
@dataclass(frozen=True)
class DatasetConfig:
    COUNTRY_CODE:     str
    IMG_HEIGHT:       int = 224
    IMG_WIDTH:        int = 224
    GIS_ROOT:         str = './GIS-Image-Stack-Processing'
    AOI_ROOT:         str = './GIS-Image-Stack-Processing/AOI/'
    PRT_ROOT:         str = './GIS-Image-Stack-Processing/AOI/Partitions'
    TARGET_TYPE:      Union[TargetType, List[TargetType]] = TargetType.FRACTION_WITH_FRESH_WATER

@dataclass(frozen=True)
class FeatureConfig:
    FEATURE_LAYER:   str = 'layer4'
    BLOCK_INDEX:     int = 1
    SUB_LAYER_PART:  str = 'conv2'
    RELU:            bool = True      # Set to True to extract featuers from last (ReLU) in layer.
                                      # Ignores BLOCK_INDEX and SUB_LAYER_PART
```

The default settings shown above extract features from the final output of layer4, which has been shown to perform well. However, for experimentation, other extraction points can be specified. It's important to understand how the configuration inputs work when targeting other layers or sub-layers.

The table below provides a summary of the ResNet18 model architecture, focusing on the last two sequential layers (layer3 and layer4). The configuration structure allows users to specify the Sequential layer name (FEATURE_LAYER), the BasicBlock (BLOCK_INDEX), and the Sub-layer name (SUB_LAYER_PART).

Special Case: RELU Layer

An exception exists for the RELU sub-layer at the end of each sequential layer. If the RELU output is the desired extraction point:

- Set RELU = True.
- BLOCK_INDEX and SUB_LAYER_PART are ignored.

For all other layers, specify BLOCK_INDEX and SUB_LAYER_PART as needed, and set RELU = False.

This flexibility enables feature extraction from any desired layer or sub-layer for experimentation.[1]

[1] Feature extraction is limited to the final ReLU sub-layer of each sequential block; intermediate ReLU layers within the blocks cannot be targeted.

```
├─Sequential (layer3)                   [1, 256, 14, 14]          --
│    └─BasicBlock (0)                   [1, 256, 14, 14]          --
│    │    └─Conv2d (conv1)              [1, 256, 14, 14]          294,912
│    │    └─BatchNorm2d (bn1)           [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                 [1, 256, 14, 14]          --
│    │    └─Conv2d (conv2)              [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn2)           [1, 256, 14, 14]          512
│    │    └─Sequential (downsample)     [1, 256, 14, 14]          33,280
│    │    └─ReLU (relu)                 [1, 256, 14, 14]          --
│    └─BasicBlock (1)                   [1, 256, 14, 14]          --
│    │    └─Conv2d (conv1)              [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn1)           [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                 [1, 256, 14, 14]          --
│    │    └─Conv2d (conv2)              [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn2)           [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                 [1, 256, 14, 14]          --
├─Sequential (layer4)                   [1, 512, 7, 7]            --
│    └─BasicBlock (0)                   [1, 512, 7, 7]            --
│    │    └─Conv2d (conv1)              [1, 512, 7, 7]            1,179,648
│    │    └─BatchNorm2d (bn1)           [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                 [1, 512, 7, 7]            --
│    │    └─Conv2d (conv2)              [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn2)           [1, 512, 7, 7]            1,024
│    │    └─Sequential (downsample)     [1, 512, 7, 7]            132,096
│    │    └─ReLU (relu)                 [1, 512, 7, 7]            --
│    └─BasicBlock (1)                   [1, 512, 7, 7]            --
│    │    └─Conv2d (conv1)              [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn1)           [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                 [1, 512, 7, 7]            --
│    │    └─Conv2d (conv2)              [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn2)           [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                 [1, 512, 7, 7]            --
├─AdaptiveAvgPool2d (avgpool)           [1, 512, 1, 1]            --
├─Sequential (fc)                       [1, 5]                    --
│    └─Dropout (0)                      [1, 512]                  --
│    └─Linear (1)                       [1, 5]                    2,565
==========================================================================================
```

ResNet18 Model Configuration

The next section in the notebook is used to configure the ResNet18 model. In the `TrainingConfig` block, the user specifies the model run mode based on following enumeration.

```
class ModelMode(Enum):
    PRE_TRAINED = "Pre_Trained"
    FINE_TUNE   = "Fine_Tune"
    CHECKPOINT  = "Checkpoint"
```

The user can select one of three options:

- `ModelMode.PRE_TRAINED`: Only the batch size needs to be specified for loading data.
- `ModelMode.CHECKPOINT`: Requires a checkpoint file to be present in the `./ResNet18_CHECKPOINTS` folder. Only the batch size is used for loading data. When this mode is selected, the notebook constructs a filename based on the current training configuration settings and searches for it in the file system. For example:

  `./ResNet18_CHECKPOINTS`

  `ResNet18_4layers_5targets_lr_4e-05_bs_8_l2_0.02_do_0.4_ft_4_log_1_aug_1_trg_5__9AOI.pth`

This approach ensures consistency but can also be tedious. Allowing direct filename input could simplify the process, however, the current notebook version does not support this approach.

- `ModelMode.FINE_TUNE`: Fine-tuning utilizes additional configuration parameters in the `TrainingConfig` block. The details of fine-tuning are beyond the scope of this document.

If any issues are encountered while using `ModelMode.CHECKPOINT`, it is recommended to switch the run mode to `ModelMode.PRE_TRAINED`, in which the ResNet18 pre-trained model is used directly. The pre-trained model should produce results very similar to the fine-tuned model.


```python
@dataclass(frozen=True)
class TrainingConfig:
    OUTPUTS:           int
    VERBOSE:           bool  = True
    EPOCHS:            int   = 51
    LEARNING_RATE:     float = .00004
    BATCH_SIZE:        int   = 8
    L2_REG:            float = .02
    DROPOUT:           float = .4
    PATIENCE:          int   = 10
    NUM_WORKERS:       int   = num_workers
    FINE_TUNE_LAYERS:  int   = 4
    USE_DATA_AUG:      bool  = True
    USE_LOG1P:         bool  = True
    MODEL_MODE:        ModelMode = ModelMode.CHECKPOINT  # Specify the type of model
    LOG_DIR:           str   = "./ResNet18_LOGS_DATA"
    CHECKPOINT_DIR:    str   = "./ResNet18_CHECKPOINTS"
    CASE_STRING:       str   = "9AOI"                    # Optional: Additional case string

    # 9 AOI Values
    MEAN_STD: dict = field(default_factory=lambda: {
        'Nightlights': (0.4051, 0.3869),
        'Population':  (1.5032, 2.2611),
        'Rainfall':    (1.4170, 0.8373)
    })

    def get_checkpoint_file(self, training_string: str = "") -> str:
        case_str = training_string if training_string else self.CASE_STRING
        return f"ResNet18_{self.FINE_TUNE_LAYERS}layers_{self.OUTPUTS}targets_{case_str}.pth"

    @staticmethod
    def from_dataset_config(dataset_config: DatasetConfig):
        if isinstance(dataset_config.TARGET_TYPE, list):
            num_targets =  len(dataset_config.TARGET_TYPE)
        else:
            num_targets = 1
        return  TrainingConfig(OUTPUTS=num_targets)


# Result configurations
@dataclass(frozen=True)
class ResultsConfig:
    COMPUTE_GEOSPATIAL:  bool = True
    COMPUTE_AOI_DIST:    bool = False  # Takes time when set to True
    PLOT_TRAINING_DIR:    str = './Plots_Fine_Tuning'
    PLOT_UMAP_FEAT_DIR:   str = './Plots_Projected_Features/9AOI'
    PLOT_GEOSPATIAL_DIR:  str = './Plots_Geospatial/9AOI'
```

The section of the notebook shown below allows model fine-tuning to be configured for multiple or single regression which is only applicable when `ModelMode = ModelMode.FINE_TUNE`. Specifically, It allows the user to configure the regression head for one or more DHS survey metrics. The configuration below is set up for multiple regression using five survey metrics as target variables. To configure the regression head for single variable regression, toggle the comments between the two sections below and specify which target variable to use from the provided dictionary.

```python
#-------------------------------------------------------------------------------
# Multiple Regression

dataset_config = DatasetConfig(COUNTRY_CODE=country_code,
                               TARGET_TYPE=[TargetType.FRACTION_DPT3_VACCINATED,
                                            TargetType.FRACTION_WITH_ELECTRICITY,
                                            TargetType.MEAN_WEALTH_INDEX,
                                            TargetType.FRACTION_WITH_RADIO,
                                            TargetType.FRACTION_WITH_TV])

for target in dataset_config.TARGET_TYPE:
    print(f"Selected Target: {target_name_mapping[target]}")

desired_target_type_key = ""
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# # Or, Single Regression

# # Define a mapping between target type strings and the corresponding TargetType enums
# target_type_mapping = {
#     'dpt3'        : [TargetType.FRACTION_DPT3_VACCINATED],
#     'wealth'      : [TargetType.MEAN_WEALTH_INDEX],
#     'electricity' : [TargetType.FRACTION_WITH_ELECTRICITY],
#     'water'       : [TargetType.FRACTION_WITH_FRESH_WATER]
# }

# desired_target_type_key = 'dpt3'  # Specify which metric to use for single regression

# # Use the desired target type key to select the appropriate TARGET_TYPE
# dataset_config = DatasetConfig(
#     COUNTRY_CODE=country_code,
#     TARGET_TYPE=target_type_mapping[desired_target_type_key]
# )
#-------------------------------------------------------------------------------
```

K-Means Clustering

Further below in the notebook, after features have been extracted from ResNet18 (and/or satellite features loaded from disk), features are projected using UMAP and the resulting points are plotted in a two-dimensional plot of the UMAP projection. At this point, clustering is performed using K-Means and requires the user to specify the number of clusters which is a required input for K-means. This is typically an iterative process, but to facilitate this, a Silhouette Score is computed as a metric to iteratively determine the optimal number of clusters for a given AOI. Silhouette Scores greater than approximately 0.70 are generally considered fairly good. However, the goal is to identify the number of clusters that maximizes the score.

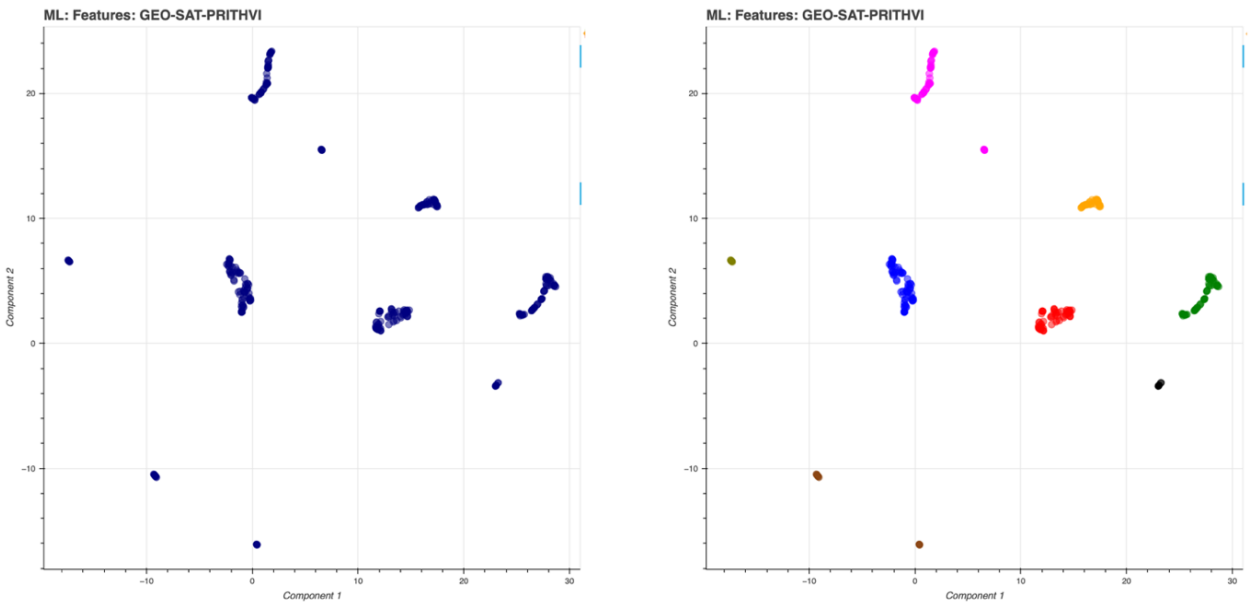
```
aoi_num_clusters = {
    'AM': 3,
    'MA': 3,
    'MB': 3,
    'ML': 8,
    'MR': 3,
    'NI': 4,
    'PK': 8,
    'SN': 3,
    'TD': 3
}

n_clusters = aoi_num_clusters.get(country_code, 0)

kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=20, max_iter=50)

cluster_labels = kmeans.fit_predict(projected_features)

print(cluster_labels.shape)
```

```
(322,)
```

```
score = silhouette_score(projected_features, cluster_labels)
print(f'Silhouette Score: {score:.2f}')
```

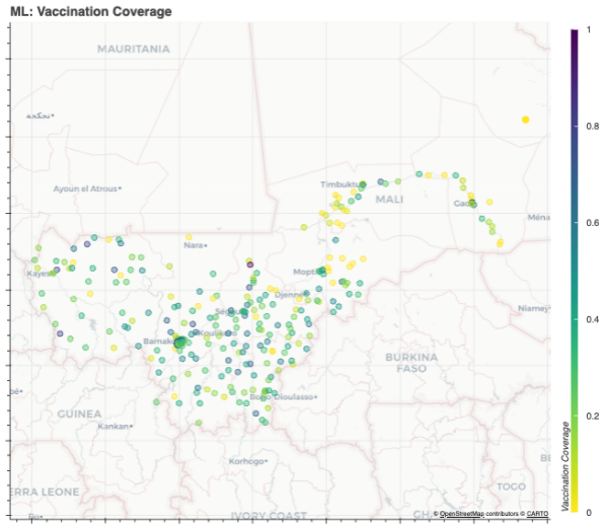
```
Silhouette Score: 0.84
```

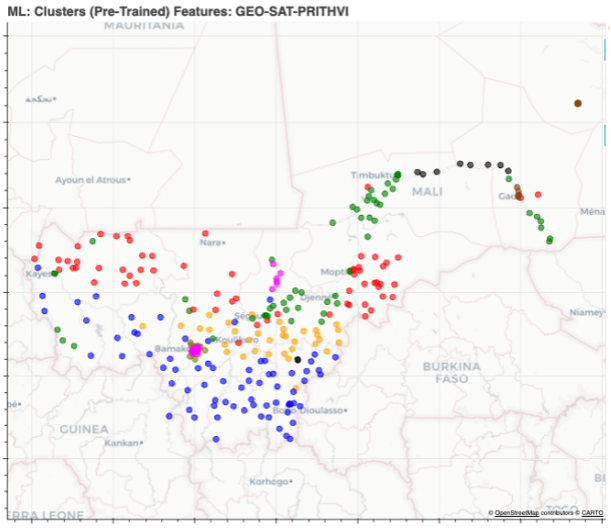
The example below displays the un-clustered UMAP projection on the left and the K-means clustering result with eight clusters on the right. This configuration achieved the highest Silhouette Score of 0.84 after testing a range of cluster numbers.

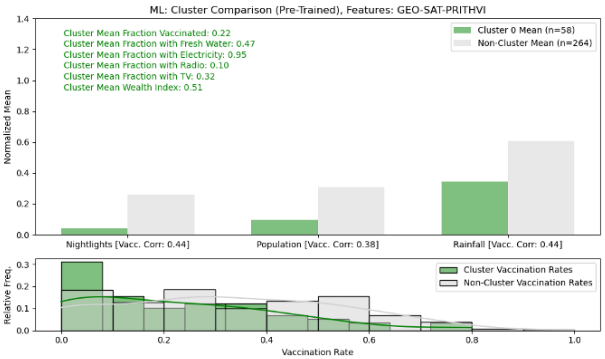


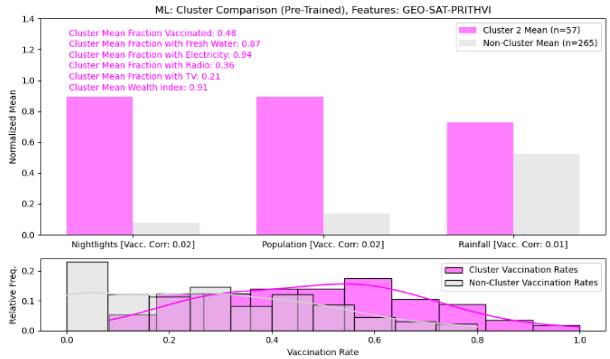
Unsupervised Clustering Results

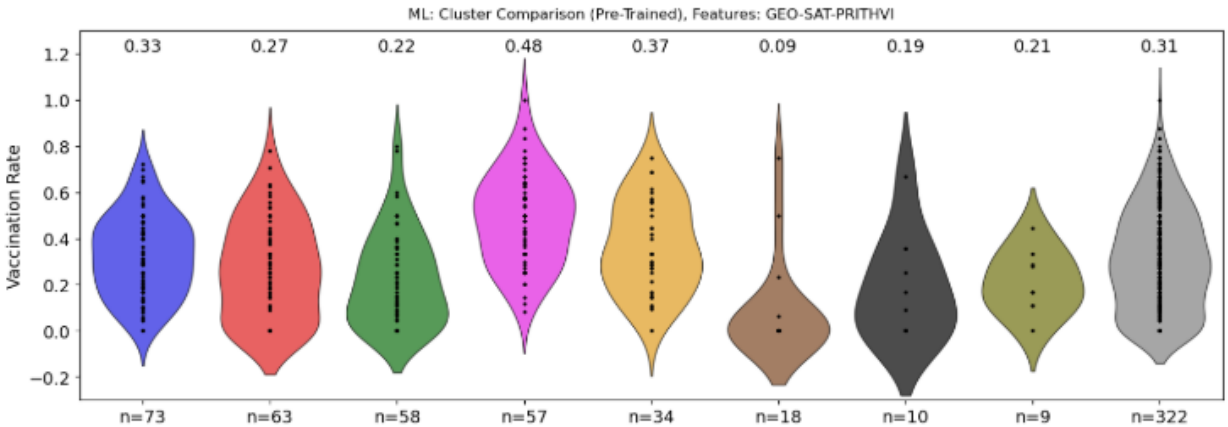
Once clustering has been performed, several plots are produced to visualize the results which include the geographic distribution of the clusters, the within-cluster distribution of vaccination rates, as well as violin plots to identify skewed distributions.







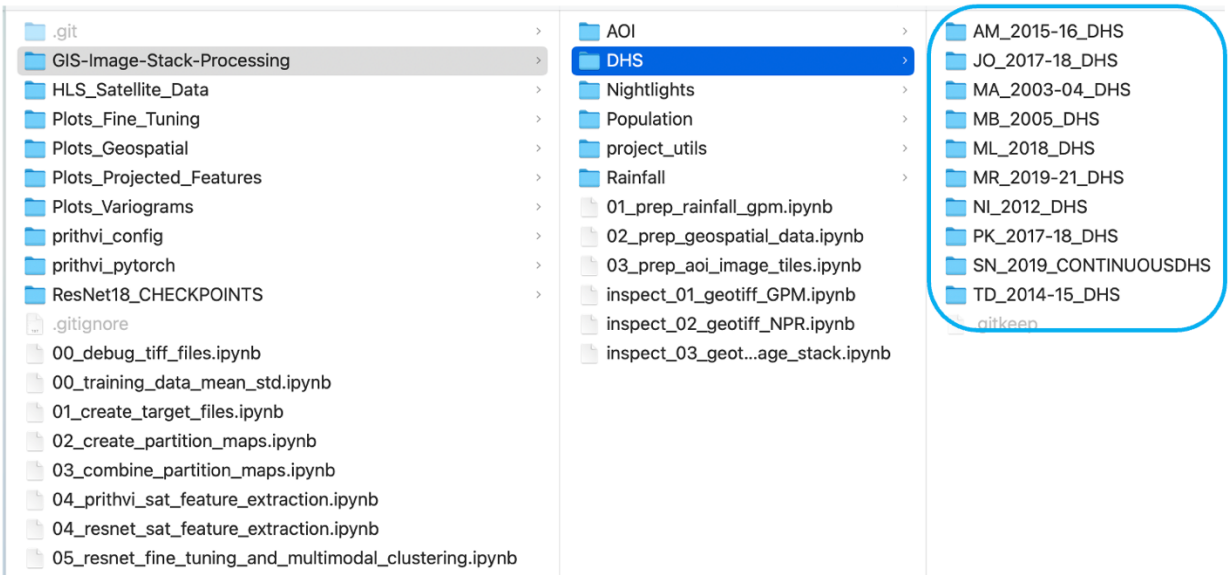




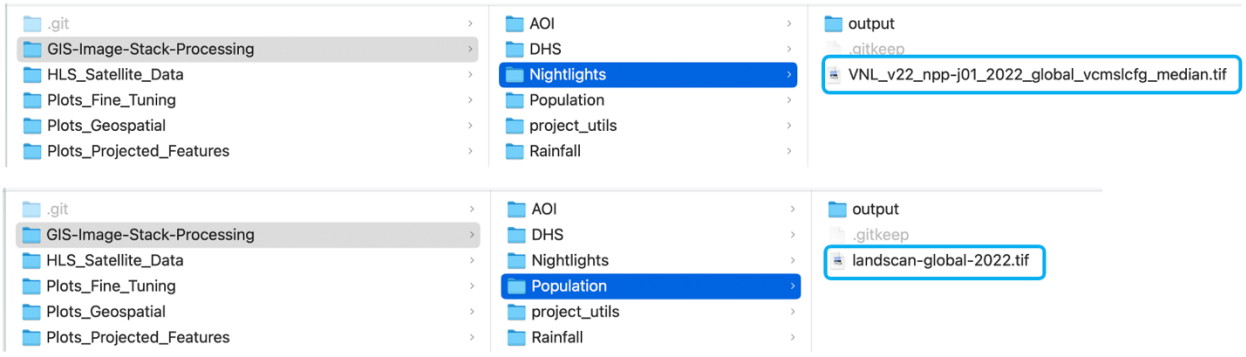
3. Project DHS and Geospatial Data

This section outlines the file system structure and organization of geospatial and DHS data. It specifies where input data should be located to ensure that all geospatial notebooks in the repository can be executed from start to finish. These steps include pre-processing geospatial data, creating AOI image tiles, and preparing data for ResNet18 processing.

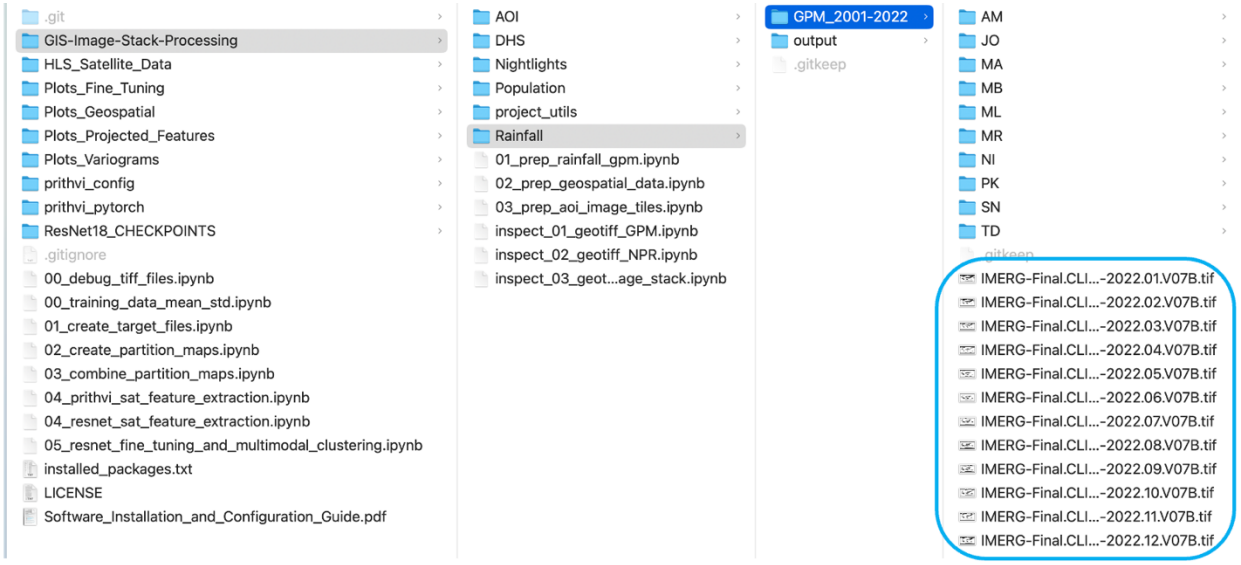
Since geospatial data preprocessing and image tile generation are one-time tasks, users can also choose to directly populate the AOI folder with pre-processed files and proceed with the main processing pipeline for unsupervised cluster analysis. The filesystem view below shows the location for AOI DHS data. An archive of this data will be available for download and should be extracted to the DHS folder.



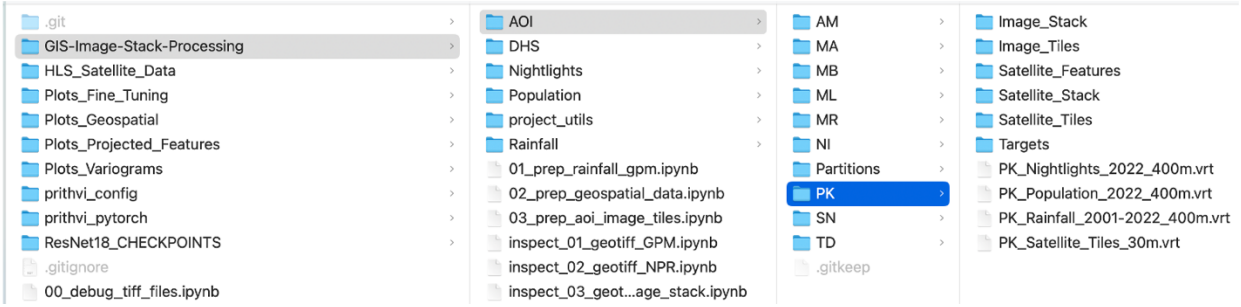
Each geospatial data type includes global source files. For Nightlights and Population, these are single, large GeoTIFF files covering the entire globe. Each file should be placed in its corresponding folder, as outlined below.



The source data for Rainfall is more complicated since it is provided as 12 monthly files. These files should be placed in the `GPM_2001-2022` folder, as shown below. The AOI folders contained in the `GPM_2001-2022` folder are produced by the first notebook (`01_prep_rainfall_gpm.ipynb`)



Once all geospatial data has been processed, each country's AOI folder will be populated as shown below. To save time and effort, an archive containing this data has been provided, eliminating the need to process everything from scratch.



Appendix: Notebook Configuration Notes

Three notebooks require the use of binary files as explained in the comments below. Therefore, the pathname to the bin folder in the Conda virtual environment should be edited in each applicable notebook to reflect the location of this folder on the filesystem.

```
01_prep_rainfall_gpm.ipynb
02_prep_geospatial_data.ipynb
03_prep_aoi_image_tiles.ipynb
```

```
8  #------------------------------------------------------------------------------------------
9  # *** IMPORTANT: SYSTEM PATH TO SET ***
10 #------------------------------------------------------------------------------------------
11 # The following path is required, as it contains GDAL binaries used for several
12 # pre-processing functions. The pathname corresponds to the Conda virtual environment
13 # created for this project (e.g., "py39-pt").
14 #
15 # Note: GDAL was adopted as a benchmark to compare the original GIS data produced by
16 # another team. However, similar functionality could be implemented using the Rasterio
17 # Python package. If Rasterio is used, it would eliminate the need for GDAL binaries
18 # and this system path specification.
19 #------------------------------------------------------------------------------------------
20
21 os.environ['PATH'] += ':/Users/billk/miniforge3/envs/py39-pt/bin/'
```

Conda virtual environment