```python
from pprint import pprint
from PIL import Image
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import glob
import xml.etree.ElementTree as ET

# Distance betwen moon and sun in AUs
D_AU = 1

# Solar irradiance
F0 = np.array([
  136.1259307,
  129.8781929,
  125.1457188,
  120.4566749,
  115.2187742,
  110.7989129,
  105.971862,
  102.2853476,
  98.83159112,
  95.00990644,
  91.72241746,
  88.63043389,
  85.44216416,
  83.09659958,
  80.7461688,
  77.99745659,
  75.43755054,
  72.53298554,
  70.30310472,
  67.71506702,
  65.53063581,
  63.51647332,
  61.49193881,
  59.39769145,
  57.24811211,
  55.56974549,
  53.96628612,
  52.39858882,
  50.94286582,
  49.55873832,
  47.99340839,
  46.35543865,
  45.11640663,
  43.75374359,
  42.46741487,
  41.1950428,
  39.93375405,
  38.7480202,
  37.63257797,
  36.52968828,
  35.48372942,
  34.51571377,
  33.5041102,
  32.62925225,
  31.80035805,
  30.98128654,
  30.16775831,
  29.32709974,
  28.56074168,
  27.8298174,
  27.0453247,
  26.30808675,
  25.51810387,
  24.75010497,
  24.00573968,
  23.24760491,
  22.51761852,
  21.78398871,
  21.06792047,
  20.39822233,
  19.7458807,
  19.11661541,
  18.44061437,
  17.83250529,
  17.26068394,
  16.65126453,
  16.11545704,
  15.61912435,
  15.1210474,
  14.62910738,
  14.16359209,
  13.72237684,
  13.31430194,
  12.94713935,
  12.56233275,
  12.18239943,
  11.79722098,
  11.38810049,
  11.04636914,
  10.71621297,
  10.38904988,
  10.06620698,
  9.753295821,
  9.46418631,
  9.201075776,
  8.960974818,
  8.732115834,
  8.508712424,
  8.28861478,
  8.070068082,
  7.850866176,
  7.629585176,
  7.417896212,
  7.21399149,
  7.014245694,
  6.819995994,
  6.637200746,
  6.463212542,
  6.291676014,
  6.122400975,
  5.952327234,
  5.785907458,
  5.631916792,
  5.48221029,
  5.338864421,
  5.183886388,
  5.053359936,
  4.941756508,
  4.835098184,
  4.719922707,
  4.619729215,
  4.511137419,
  4.407240202,
  4.306184976,
  4.210413629,
  4.117013411,
  4.012368768,
  3.918726643,
  3.824014432,
  3.725826304,
  3.646586732,
  3.564719937,
  3.488199195,
  3.397463341,
  3.32250234,
  3.262984894,
  3.190955311,
  3.122692223,
  3.056477464,
  2.991274348,
  2.926566072,
  2.864612339,
  2.802940836,
  2.743157021,
  2.685370618,
  2.628641884,
  2.571929704,
  2.517226294,
  2.465127643,
  2.414375576,
  2.365285234,
  2.316701141,
  2.26923212,
  2.222564505,
  2.178496705,
  2.135290025,
  2.092826765,
  2.051565701,
  2.010893773,
  1.971470582,
  1.932492639,
  1.893925453,
  1.853239032,
  1.814419696,
  1.780829606,
  1.751599126,
  1.715922793,
  1.680125966,
  1.647791753,
  1.621454182,
  1.593640531,
  1.560460708,
  1.532378246,
  1.507178355,
  1.480349348,
  1.454525518,
  1.426003985,
  1.40026592,
  1.376814112,
  1.351395724,
  1.327241488,
  1.303320437,
  1.279240078,
  1.255715058,
  1.232621586,
  1.209534773,
  1.186777237,
  1.163774025,
  1.141839466,
  1.121354795,
  1.102697582,
  1.084984542,
  1.06779729,
  1.050654559,
  1.034116451,
  1.018239678,
  1.003106371,
  0.987228033,
  0.971082552,
  0.954532246,
  0.938549781,
  0.922761605,
  0.90746215,
  0.892772367,
  0.876952832,
  0.86169586,
  0.846904043,
  0.832961745,
  0.820193322,
  0.808495532,
  0.796418017,
  0.784036511,
  0.771772032,
  0.760169612,
  0.74902997,
  0.737997332,
  0.727055348,
  0.716477866,
  0.704633464,
  0.691770452,
  0.681177697,
  0.668685204,
  0.6563386,
  0.643784606,
  0.630929839,
  0.618670348,
  0.605670184,
  0.593191697,
  0.582320158,
  0.571630629,
  0.561438106,
  0.551831735,
  0.542986524,
  0.534529199,
  0.526707332,
  0.518722109,
  0.511109087,
  0.50373316,
  0.496221855,
  0.489530981,
  0.482582186,
  0.475974536,
  0.469794569,
  0.463575699,
  0.458286546,
  0.452850271,
  0.447197638,
  0.441572082,
  0.43580287,
  0.430755766,
  0.425717099,
  0.420589447,
  0.41588213,
  0.410468477,
  0.405233536,
  0.399887123,
  0.394668014,
  0.389642973,
  0.384580319,
  0.379611238,
  0.374544041,
  0.369613524,
  0.364863435,
  0.360132602,
  0.355533758,
  0.350967069,
]).reshape(256,1,1)

class utils:
    @staticmethod
    def _extract_sequence_numbers(file_path):
        """
        Extracts the text content of <sequence_number> elements from
an XML file.

        Args:
            file_path (str): The path to the XML file.

        Returns:
            list: A list of text contents from <sequence_number>
elements.
        """
        # Parse the XML file
        tree = ET.parse(file_path)
        root = tree.getroot()

        # Find all sequence_number elements
        sequence_numbers =
root.findall('.//{http://pds.nasa.gov/pds4/pds/v1}elements')

        # Extract the text content from each <sequence_number> element
        return [int(sequence_number.text) for sequence_number in
```

```
sequence_numbers]
    @staticmethod
    def _find_xml_files(base_path):
        """
        Finds all XML files matching the pattern
/data/calibrated/*/*.xml within the given base path.

        Args:
            base_path (str): The base directory path where the search
begins.

        Returns:
            list: A list of paths to the matching XML files.
        """
        # Define the search pattern
        pattern = os.path.join(base_path, 'data', 'calibrated', '*',
'*.xml')

        # Use glob to find all files matching the pattern
        matching_files = glob.glob(pattern)
        matching_files.sort()
        return matching_files

    @staticmethod
    def _find_qub_files(base_path):
        """
        Finds all .qub files matching the pattern
/data/calibrated/*/*.qub within the given base path.

        Args:
            base_path (str): The base directory path where the search
begins.

        Returns:
            list: A list of paths to the matching .qub files.
        """
        # Define the search pattern
        pattern = os.path.join(base_path, 'data', 'calibrated', '*',
'*.qub')

        # Use glob to find all files matching the pattern
        matching_files = glob.glob(pattern)
        matching_files.sort()
        return matching_files
    @staticmethod
    def _get_image_array(qub_path,shape):
        """
        Args:
            qub_path : path to the .qub file
            shape : shape of the image of form (channels,height,width)
        returns:
            a numpy array reshaped in the shape provided
        """
        with open(qub_path, 'rb') as f:
            img = np.frombuffer(f.read(),
dtype=np.float32).reshape(*shape)
        return img

    @staticmethod
    def get_image(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A list of images with the radiance resized and resahped.
To visualize it you need to do min max scaling and stuff
        """
        xml_files = utils._find_xml_files(base_path)
        image_files = utils._find_qub_files(base_path)
        shapes = [utils._extract_sequence_numbers(xml_file) for
xml_file in xml_files]
        images = [utils._get_image_array(qub_path,shape) for
qub_path,shape in zip(image_files,shapes)]
        return images
    @staticmethod
    def _read_misc_files_into_df(file_path):
        with open(file_path) as f:
            data = f.read()
        rows = data.split('\n')
        parsed_list = [' '.join(row.split()).split() for row in rows]
        return pd.DataFrame(parsed_list)

    @staticmethod
    def _get_misc_files(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A dictionary of miscellaneous files converted to the
dataframe
        """
        pattern = os.path.join(os.path.join(base_path,
'miscellaneous', 'calibrated', '*', '*.*'))
        matching_files = glob.glob(pattern)
        matching_files.sort()
        dfs = {}
        for file_path in matching_files:
            print(file_path)
            _, file_extension = os.path.splitext(file_path)
            dfs[file_extension] =
utils._read_misc_files_into_df(file_path).dropna()
        return dfs
    @staticmethod
    def get_misc_files(base_path):
        """
        Args:
            base_path : Base directory path to the data. Which means
the path to directory which contains the browse, data geometyr and
miscellaneous folder
        returns:
            A dictionary of miscellaneous files converted to the
dataframe
        """

        dfs = utils._get_misc_files(base_path)
        dfs['.spm'] = utils._process_spm_df(dfs['.spm'])
        dfs['.oat'] = utils._process_oat_df(dfs['.oat'])
        return dfs
    @staticmethod
    def convert_to_reflectance(data, solar_zenith_angle):
        return (np.pi * data)/(np.cos(solar_zenith_angle * np.pi /
180) * F0 * D_AU**2)

    @staticmethod
    def _process_spm_df(spm_df):
        column_names = [
            "Record type", "Physical record number", "Block length",
        #     "Year",
            "Month",
            "Date",
            "Hour",
            "Minute",
            "Second",
            "Millisec", "Satellite position X", "Satellite position
Y",
            "Satellite position Z", "Satellite velocity X-dot",
            "Satellite velocity Y-dot", "Satellite velocity Z-dot",
            "Phase angle", "Sun aspect", "Sun Azimuth",
            "Sun Elevation",
        ]

        # Assign the new column names
        spm_df.columns = column_names
        # Convert the columns to the appropriate data types
        spm_df["Record type"] = spm_df["Record type"].astype(str)
        spm_df["Physical record number"] = spm_df["Physical record
number"].astype(np.int32)
        spm_df["Block length"] = spm_df["Block
length"].astype(np.int32)
        # spm_df['Year'] = spm_df['Year'].astype(np.int32)
        spm_df['Month'] = spm_df['Month'].astype(np.int32)
        spm_df['Date'] = spm_df['Date'].astype(np.int32)
        spm_df['Hour'] = spm_df['Hour'].astype(np.int32)
        spm_df['Minute'] = spm_df['Minute'].astype(np.int32)
        spm_df['Second'] = spm_df['Second'].astype(np.int32)
        spm_df['Millisec'] = spm_df['Millisec'].astype(np.int32)
        # For "Time in UTC", assuming it is in the format
'YYYYMMDDHHMMSS', convert to datetime

        spm_df["Satellite position X"] = spm_df["Satellite position
X"].astype(np.float32)
        spm_df["Satellite position Y"] = spm_df["Satellite position
Y"].astype(np.float32)
        spm_df["Satellite position Z"] = spm_df["Satellite position
Z"].astype(np.float32)
        spm_df["Satellite velocity X-dot"] = spm_df["Satellite
velocity X-dot"].astype(np.float32)
        spm_df["Satellite velocity Y-dot"] = spm_df["Satellite
velocity Y-dot"].astype(np.float32)
        spm_df["Satellite velocity Z-dot"] = spm_df["Satellite
velocity Z-dot"].astype(np.float32)
        spm_df["Phase angle"] = spm_df["Phase
angle"].astype(np.float32)
        spm_df["Sun aspect"] = spm_df["Sun aspect"].astype(np.float32)
        spm_df["Sun Azimuth"] = spm_df["Sun
Azimuth"].astype(np.float32)
        spm_df["Sun Elevation"] = spm_df["Sun
Elevation"].astype(np.float32)
        # spm_df["Orbit Limb Direction"] = spm_df["Orbit Limb
Direction"].astype(int)
        return spm_df
    @staticmethod
    def _process_oat_df(oat_df):
        columns_names = [
            "Record type",
            "Physical record number in this file",
            "Block length in bytes",
            "Month",
            "Date",
            "Hour",
            "Minute",
            "Second",
            "Millisec",
            "Lunar Position X (kms) - J2000 Earth Centre Frame",
            "Lunar Position Y (kms) - J2000 Earth Centre Frame",
            "Lunar Position Z (kms) - J2000 Earth Centre Frame",
            "Satellite position X (kms) - Note-3",
            "Satellite position Y (kms) - Note-3",
            "Satellite position Z (kms) - Note-3",
            "Satellite velocity X-dot (kms/sec) - Note-3",
            "Satellite velocity Y-dot (kms/sec) - Note-3",
            "Satellite velocity Z-dot (kms/sec)  - Note-3",
            "Altitude Inertial Q1",
            "Altitude Inertial Q2",
            "Altitude Inertial Q3",
            "Altitude Inertial Q4",
            "Earth Fixed IAU frame Q1",
            "Earth Fixed IAU frame Q2",
            "Earth Fixed IAU frame Q3",
            "Earth Fixed IAU frame Q4"
            "Lunar Fixed IAU frame Q1",
            "Lunar Fixed IAU frame Q2",
            "Lunar Fixed IAU frame Q3",
            "Lunar Fixed IAU frame Q4",
            "Latitude of sub-satellite point (deg)",
            "Longitude of sub-satellite point (deg)",
            "Solar Azimuth",
            "Solar Elevation",
            "Latitude  (deg)",
            "Longitude (deg)",
            "Satellite altitude (kms)",
            "Angle between +Roll and Velocity Vector",
            "Eclipse Status - Note-4",
            "Emission Angle",
            "Sun Angle w.r.t -ve Yaw (Phase angle)",
            "Angle between +Yaw and Nadir",
            "Slant Range (Km)",
            "Orbit No",
            "Solar Zenith Angle",
            "Angle between Payload FoV axis and velocity vector",
            "X  (yaw) angle",
            "Y  (roll) angle",
            "Z(pitch) angle",
        ]
        oat_df.columns = columns_names
        oat_df.iloc[:,1:9] = oat_df.iloc[:,1:9].astype(np.int32)
        oat_df.iloc[:,9:42] = oat_df.iloc[:,9:42].astype(np.float32)
        oat_df.iloc[:,42] = oat_df.iloc[:,42].astype(np.int32)
        oat_df.iloc[:,43:] = oat_df.iloc[:,43:].astype(np.float32)
        return oat_df
```

```
# data path should be root directory of the bundle
data_path = "ENTER_THE_PATH_TO_ROOT_DIRECTORY_TO_BUNDLE"
images = utils.get_image(data_path)
image = images[0]
misc_dfs = utils.get_misc_files(data_path)
oat_df = misc_dfs['.oat']
mean_zenith_angle = oat_df.loc[:,'Solar Zenith Angle'].mean()
refletance_image =
utils.convert_to_reflectance(image,mean_zenith_angle)
# reflectance image is the desired output
```

Torch Dataset

```
import os
import numpy as np
import torchdata.datapipes as dp
from torch.utils.data import DataLoader

class SlidingWindowDataPipe(dp.iter.IterDataPipe):
    def __init__(self, image_paths, window_size, step_size):
        super().__init__()
        self.image_paths = image_paths
        self.window_size = window_size
        self.step_size = step_size

    def __iter__(self):
        for image_path in self.image_paths:
            shapes =
SlidingWindowDataPipe.get_image_meta_data(image_path)
            for shape in shapes:
                _,img_height, img_width = shape
                # Sliding window
                for i in range(0, img_height - self.window_size + 1,
self.step_size):
                    if i + self.window_size > img_height:
                        continue
                    patch =
SlidingWindowDataPipe.get_partial_image_from_height(image_path,i,self.
window_size)
                    yield patch
    @staticmethod
    def _read_partial_data_of_given_height(qub_path,image_height,
image_width,row = 0, height = 250):
        image = []
        data_count = image_width * height
        channel_size = image_height * image_width
        with open(qub_path,'rb') as f:
            for channel_idx in range(256):
                offset = channel_idx * channel_size * 4 + row * 
image_width  # float32 has 4 bytes
                f.seek(offset)
                channel_data = np.fromfile(f, dtype=np.float32, 
count=data_count)
                image.append(channel_data.reshape((1,height, 
image_width)))
        return np.vstack(image)
    @staticmethod
    def get_partial_image_from_height(base_path, row, height):
        xml_files = utils.find_xml_files(base_path)
        image_files = utils.find_qub_files(base_path)
        shapes = [utils.extract_sequence_numbers(xml_file) for 
xml_file in xml_files]
        images = 
[utils._read_partial_data_of_given_height(qub_path,shape[1], shape[2], 
row, height) for qub_path,shape in zip(image_files,shapes)]
        return images
    @staticmethod
    def get_image_meta_data(base_path):
        xml_files = utils.find_xml_files(base_path)
        shapes = [utils.extract_sequence_numbers(xml_file) for 
xml_file in xml_files]
        return shapes


data_paths = [
    "ENTER_LIST_OF_PATHS",
]

datapipe = SlidingWindowDataPipe(
    image_paths=data_paths,
    window_size=250,
    step_size=250
)
dataloader = DataLoader(datapipe, batch_size=12, shuffle=True)
dataloader_iter = iter(dataloader)
```

Example Usage of dataset

```
dataloader_iter.__next__()
```