1 Cyprus Earth Observation Data Cube

1.1 Available satellite products

Product	Code name
MOD13Q1 - MODIS/Terra Vegetation Indices 16-Day L3 Global 250 m	CYPRUS_MOD13Q1
MYD13Q1 - MODIS/Terra Vegetation Indices 16-Day L3 Global 250 m	CYPRUS_MYD13Q1
MOD11A1 - MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 1 km	CYPRUS_MOD11A1
MYD11A1 - MODIS/Terra Land Surface Temperature/Emissivity Daily L3 Global 1 km	CYPRUS_MYD11A1
MOD14A1 - MODIS/Terra Thermal Anomalies/Fire Daily L3 Global 1 km	CYPRUS_MOD14A1
MYD14A1 - MODIS/Terra Thermal Anomalies/Fire Daily L3 Global 1 km	CYPRUS_MYD14A1
MOD15A2H - MODIS/Terra Leaf Area Index/FPAR 8-Day L4 Global 500 m	CYPRUS_MOD15A2H
MYD15A2H - MODIS/Terra Leaf Area Index/FPAR 8-Day L4 Global 500 m	CYPRUS_MYD15A2H
MOD17A2H - MODIS/Terra Gross Primary Productivity 8-Day L4 Global 500 m	CYPRUS_MOD17A2H
MYD17A2H - MODIS/Terra Gross Primary Productivity 8-Day L4 Global 500 m	CYPRUS_MYD17A2H
MOD16A2 - MODIS/Terra Net Evapotranspiration 8-Day L4 Global 500 m	CYPRUS_MOD16A2
MYD16A2 - MODIS/Terra Net Evapotranspiration 8-Day L4 Global 500 m	CYPRUS_MYD16A2
MOD44B - MODIS/Terra Vegetation Continuous Fields Yearly L3 Global 250	CYPRUS_MOD44B
Sentinel-1 C-Band Synthetic Aperture Radar Ground Range Detected	CYPRUS_Sentinel1_GRD
Sentinel-2 Multi-Spectral Instrument	CYPRUS_Sentinel2_MSI_L2A

1.1.1 Important information

All of the products (and their corresponding datasets) index in Cyprus Earth Observation Data Cube (CEODC) are reprojected to EPSG:32636 (https://epsg.io/32636).

1.1.2 How to load a time-series?

Import libraries

[1]:
```
import datacube
```

Establish a connection/app on CEODC

[2]:
```
dc = datacube.Datacube(app="load_ts")
```

Set starting and ending date for your study

[3]:
```
start_time, end_time = "2024-01-01","2024-01-30"
```

Set up a query to filet datasets

[4]:
```
#In every search procedure in CEODC's database user have to indicate:
#output_crs (e.g., EPSG:32636),
#resolution (-native resolution, native resolution), time.
#Dask chunks can be used for parallel processing.
query = {"output_crs":("EPSG:32636"),"resolution":(-250,250),
         "time":(start_time,end_time),"dask_chunks":{}}
```

1.1.3 Load images according to query

[23]:
```
#Product defines the product which under, datasets will be loaded
#**query_s2 pass query's arguments
#measurements defines certain layers/bands of a product's datasets to be laoded
data = dc.load(product="CYPRUS_Sentinel2_MSI_L2A", **query, measurements =
 ↪["B02","B03","B04","B05","B06","B07","B08","B8A","B09","B11","SCL"])
```

[6]:
```
data
```

[6]:
```
<xarray.Dataset>
Dimensions:      (time: 6, y: 781, x: 1241)
Coordinates:
  * time         (time) datetime64[ns] 2024-01-03 2024-01-08 … 2024-01-28
  * y            (y) float64 3.985e+06 3.985e+06 3.985e+06 … 3.79e+06 3.79e+06
  * x            (x) float64 3.999e+05 4.001e+05 … 7.096e+05 7.099e+05
    spatial_ref  int32 32636
Data variables:
    B02          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
```

```
    B03          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B04          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B05          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B06          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B07          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B08          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B8A          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B09          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
    B11          (time, y, x) uint16 dask.array<chunksize=(1, 781, 1241),
meta=np.ndarray>
Attributes:
    crs:           EPSG:32636
    grid_mapping:  spatial_ref
```

1.1.4 Another way to find (not load) datasets for a desired time period and product

[7]:
```
datasets = dc.find_datasets(
    product='CYPRUS_Sentinel2_MSI_L2A',
    time=(start_time, end_time)
)
```

[10]:
```
print(datasets[0:4])
```

```
[Dataset <id=5a6dada8-3560-4537-8724-8b7a5ab248f7
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024/T36SWD/dataset_definition_S2A_
MSIL2A_20240108T083331_N0510_R021_T36SWD_20240108T114557_indexed.yaml>, Dataset
<id=d150caa5-b9cc-41c6-a21e-2652c17b29fa product=CYPRUS_Sentinel2_MSI_L2A locati
on=file:///app/data/definitions_preparer/dataset_definitions/sentinel_two_not_in
dexed/2024/T36SWD/dataset_definition_S2B_MSIL2A_20240103T083249_N0510_R021_T36SW
D_20240103T100512_indexed.yaml>, Dataset
<id=9b7e9300-280c-4700-859e-7fba4ed9b0d8 product=CYPRUS_Sentinel2_MSI_L2A locati
on=file:///app/data/definitions_preparer/dataset_definitions/sentinel_two_not_in
dexed/2024/T36SXE/dataset_definition_S2B_MSIL2A_20240103T083249_N0510_R021_T36SX
E_20240103T100512_indexed.yaml>, Dataset
<id=ec870556-5ae9-475d-81b7-14212b7d45fc product=CYPRUS_Sentinel2_MSI_L2A locati
on=file:///app/data/definitions_preparer/dataset_definitions/sentinel_two_not_in
dexed/2024/T36SXE/dataset_definition_S2A_MSIL2A_20240108T083331_N0510_R021_T36SX
E_20240108T114557_indexed.yaml>]
```

1.1.5 How to filter datasets of a product?

Filter Sentinel-2 images by tile

[14]:
```
from collections import defaultdict
from datetime import datetime

datasets_by_month = defaultdict(list)
for ds in datasets:
    # Extract the month
    month = datetime.fromisoformat(
        ds.metadata_doc['properties']['datetime']).strftime("%Y-%m")
    # Check if 'T36SVD' is in the dataset path and group only those datasets
    if 'T36SVD' in str(ds.local_path):
        datasets_by_month[month].append(ds)

# Select the dataset with the least cloud cover for each month
best_datasets = []
for month, monthly_datasets in datasets_by_month.items():
    # Sort by cloud cover
    best_dataset = min(
        monthly_datasets,
        key=lambda ds: ds.metadata_doc['properties'].get('odc:cloud_cover', 100)
    )
    best_datasets.append(best_dataset)
```

[15]:
```
best_datasets
```

[15]:
```
[Dataset <id=afe23b99-acea-4fbe-8c40-c1877ef4d7cd
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024/T36SVD/dataset_definition_S2A_
MSIL2A_20240128T083211_N0510_R021_T36SVD_20240128T113252_indexed.yaml>]
```

Filter Sentinel-2 images by cloud coverage

[35]:
```
def get_cloud_cover(ds):
    properties = ds.metadata_doc["properties"]
    return properties.get("eo:cloud_cover") or properties.get("odc:
 ↪cloud_cover") or 100

filtered_datasets = [ds for ds in datasets if get_cloud_cover(ds) <= 15]
```

[36]:
```
filtered_datasets
```

[36]:
```
[Dataset <id=90fb7974-105d-42d8-a3c7-773c53eaba38
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024/T36SWD/dataset_definition_S2A_
MSIL2A_20240128T083211_N0510_R021_T36SWD_20240128T113252_indexed.yaml>,
 Dataset <id=afe23b99-acea-4fbe-8c40-c1877ef4d7cd
```

```
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024/T36SVD/dataset_definition_S2A_
MSIL2A_20240128T083211_N0510_R021_T36SVD_20240128T113252_indexed.yaml>,
 Dataset <id=feb46e39-2744-43b0-acdb-5c07d59cadc6
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024.bckp/T36SWD/dataset_definition
_S2A_MSIL2A_20240128T083211_N0510_R021_T36SWD_20240128T113252_indexed.yaml>,
 Dataset <id=34271600-5a20-4bfa-94b4-733b3ef71000
product=CYPRUS_Sentinel2_MSI_L2A location=file:///app/data/definitions_preparer/
dataset_definitions/sentinel_two_not_indexed/2024.bckp/T36SVD/dataset_definition
_S2A_MSIL2A_20240128T083211_N0510_R021_T36SVD_20240128T113252_indexed.yaml>]
```

1.1.6 Band normalization (mandatory)

[27]:
```
data["B08"] = data["B08"] / 10000
data["B04"] = data["B04"] / 10000
```

1.1.7 How to calculate a spectral index? (NDVI example)

[28]:
```
data["NDVI"] = (data["B08"] - data["B04"]) / (data["B08"] + data["B04"])
```

1.1.8 Clipping

Set up a bbox in EPSG:32636

[29]:
```
bbox = (491199.8528,3842665.0553,494668.2160,3847651.8725)
```

[30]:
```
data_clipped = data.where((data.x >= bbox[0]) & (data.x <= bbox[2]) &
                          (data.y>= bbox[1]) & (data.y <= bbox[3]), drop=True)
```

1.1.9 Masking using Sentinel-2 Scene Classification band

[32]:
```
def apply_scl_mask(data):
    # SCL values to keep (4: vegetation, 5: bare soils, 6: water)
    # For more info refer to:
    #https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/
 ↪scene-classification/
    valid_pixels = (data["SCL"] == 4) | (data["SCL"] == 5) | (data["SCL"] == 6)
    return data.where(valid_pixels, drop=False)

data_clipped = apply_scl_mask(data_clipped)
```

[ ]:
```

```

0.0.1 pyLARDA

[3]:
```
import pyLARDA
import matplotlib.pyplot as plt
import datetime

larda = pyLARDA.LARDA('remote', uri='http://172.16.63.43:7979')
campaign = 'caro_limassol'

larda.connect(campaign,build_lists=False)

system = 'POLLYNET'
parameter = 'attbsc1064'

dt_begin = datetime.datetime(2024,10,24,0,0) ## has to be a datetime object -␣
 ↪no string
dt_end = datetime.datetime(2024,10,25,0,0)
h = [0,6000]
dataset = larda.read(system,parameter,[dt_begin, dt_end],h) ## data-container␣
 ↪(dict)

if dt_begin.strftime('%Y%m%d') == dt_end.strftime('%Y%m%d'):
    date_str = dt_begin.strftime('%Y%m%d')
else:
    date_str = '{}-{}'.format(dt_begin.strftime('%Y%m%d'),dt_end.
 ↪strftime('%Y%m%d'))

fig,ax = pyLARDA.Transformations.plot_timeheight2(dataset, z_converter='log')
plt.title(f'{campaign}, {date_str}')

fig.savefig(f'./{date_str}_{campaign}_{system}_{parameter}.png')
fig.show()
```

```
>> LARDA initialized. Documentation available at https://lacros-
tropos.github.io/larda-doc/
campaign list: caro_limassol, lacros_cycare

The data from this campaign is provided by larda without warranty and liability.
```

```
Before publishing check the data license and contact the principal investigator.
Detailed information might be available using `larda.description('system',
'parameter')`.

CEILOMETER ['beta', 'cbh', 'pbl', 'rc', 'sky_cond']
HATPROg5bin ['BLH', 'BRT', 'CBH', 'HKD_quality', 'HKD_status', 'HPC', 'IRT',
'MET_T', 'MET_p', 'MET_rH', 'TPB', 'TPC', 'iwv', 'lwp']
MIRA ['LDRg', 'LDRspec', 'SNRCorFac', 'SNRco', 'SNRg', 'VELg', 'Ze', 'Zg',
'Zmie', 'Zspec', 'noise_co', 'noiseco', 'noisecx', 'npw1', 'rc', 'sw']
PARSIVEL ['Z', 'n_particles', 'rainrate']
POLLYNET ['CLASS', 'CLASSv2', 'attbsc1064', 'attbsc355', 'attbsc532',
'attbsc532NR', 'qang532_1064', 'qang532_1064v2', 'qbsc1064', 'qbsc1064v2',
'qbsc532', 'qflag532v2', 'qpardepol532', 'qpardepol532v2', 'rh', 'voldepol355',
'voldepol532', 'wvmr']
SNOOPY ['CN', 'R2', 'VEL', 'VELraw', 'advection_vel', 'beta', 'beta_raw',
'u_vel', 'v_vel', 'wind_direction']

22.0MB [00:01, 19.7MB/s]
```