In [1]:
```
from odc_gee.earthengine import Datacube
import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import rasterio as rio
import boto3
import pyspatialml
from pyspatialml import Raster
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from botocore import UNSIGNED
from botocore.client import Config
import tempfile
from sklearn.preprocessing import StandardScaler
import requests
import tempfile
from sklearn.model_selection import cross_validate, KFold
```

In [2]:
```
with rio.open('template.tif') as template:
    print("CRS: " + str(template.crs))
    print("Bounds: " + str(template.bounds))
```

```
CRS: EPSG:4326
Bounds: BoundingBox(left=-180.0, bottom=-90.0, right=180.0, top=90.0)
```

In [3]:
```
notebook_name="harvest-final"
dc = Datacube(app=notebook_name)
```

In [4]:
```
dc.list_products()
```

Out[4]:

	name	description	license	default_crs	default_resolution
name					
MOD09A1	MOD09A1	The MOD09A1 V6.1 product provides an estimate ...	None	PROJCRS["MODIS Sinusoidal",BASEGEOGCRS["WGS 84...	(-500, 500)
MOD11A2	MOD11A2		None	PROJCRS["MODIS Sinusoidal",BASEGEOGCRS["WGS 84...	(-1000, 1000)

In [6]:
```
dc.list_measurements().loc['MOD09A1']
```

Out[6]:

measurement	name	dtype	units	nodata	ali
sur_refl_b04	sur_refl_b04	int16		-32768	[surface_reflectance_for_band_4, sur_refl_b
sur_refl_b07	sur_refl_b07	int16		-32768	[surface_reflectance_for_band_7, sur_refl_b
QA	QA	uint32		0	[surface_reflectance_500m_band_quality_cont
SolarZenith	SolarZenith	int16	Degrees	-32768	[modis_solar_zenith_angle, solarze
ViewZenith	ViewZenith	int16	Degrees	-32768	[modis_view_zenith_angle, viewze
RelativeAzimuth	RelativeAzimuth	int16	Degrees	-32768	[modis_relative_azimuth_angle, relativeazim
StateQA	StateQA	uint16		0	[surface_reflectance_500m_state_flags, stat
DayOfYear	DayOfYear	uint16		0	[julian_day_of_the_year_for_the_pixel, dayof

In [7]:
```
dc.list_measurements().loc['MOD11A2']
```

Out[7]:

measurement	name	dtype	units	nodata	aliases	s
LST_Day_1km	LST_Day_1km	uint16	Kelvin	0	[day_land_surface_temperature, lst_day_1km]	
QC_Day	QC_Day	uint8		0	[daytime_lst_quality_indicators, qc_day]	
Day_view_time	Day_view_time	uint8	Hours	0	[local_time_of_day_observation, day_view_time]	
Day_view_angl	Day_view_angl	uint8	Degrees	0	[view_zenith_angle_of_day_observation, day_vie...	
LST_Night_1km	LST_Night_1km	uint16	Kelvin	0	[night_land_surface_temperature, lst_night_1km]	
QC_Night	QC_Night	uint8		0	[nighttime_lst_quality_indicators, qc_night]	
Night_view_time	Night_view_time	uint8	Hours	0	[local_time_of_night_observation, night_view_t...	
Night_view_angl	Night_view_angl	uint8	Degrees	0	[view_zenith_angle_of_night_observation, night...	
Emis_31	Emis_31	uint8		0	[band_31_emissivity, emis_31]	
Emis_32	Emis_32	uint8		0	[band_32_emissivity, emis_32]	
Clear_sky_days	Clear_sky_days	uint8		0	[days_in_clear_sky_conditions, clear_sky_days]	
Clear_sky_nights	Clear_sky_nights	uint8		0	[nights_in_clear_sky_conditions, clear_sky_nig...	

In [8]:
```
product="MOD09A1"
latitude = (-90.0, 90.0)
longitude = (-180.0, 180.0)
time=('2018-01-01', '2022-01-31')
measurements=['sur_refl_b04', 'sur_refl_b07']
output_crs='EPSG:4326'
```

In [9]:
```
mod09 = dc.load(
        product=product,
        lat=latitude,
        lon=longitude,
        time=time,
        output_crs= output_crs,
        measurements=measurements)

mod09
```

Out[9]: xarray.Dataset

► Dimensions: (**time**: 188, **latitude**: 2, **longitude**: 2)

▼ Coordinates:

time	(time)	datetime64[ns]	2018-01-01 ... 2...
latitude	(latitude)	float64	250.0 -250.0
longitude	(longitude)	float64	-250.0 250.0
spatial_ref	()	int32	4326

▼ Data variables:

sur_refl_b04	(time, latitude, longitude)	int16	-32768 -32768 ....
sur_refl_b07	(time, latitude, longitude)	int16	-32768 -32768 ....

▼ Attributes:

crs :	EPSG:4326
grid_mapping :	spatial_ref

In [10]:
```
product="MOD11A2"
latitude=(-90.0, 90.0)
longitude=(-180.0,180.0)
time=('2022-01-01', '2022-02-28')
measurements=['LST_Day_1km', 'LST_Night_1km']
output_crs = 'EPSG:4326'
```

In [11]:
```
mod11 = dc.load(
        product=product,
        lat=latitude,
        lon=longitude,
        time=time,
        measurements= measurements,
        output_crs= output_crs
)
mod11
```

Out[11]: xarray.Dataset

► Dimensions: (**time**: 8, **latitude**: 1, **longitude**: 2)

▼ Coordinates:

time	(time)	datetime64[ns]	2022-01-01 ... 2...
latitude	(latitude)	float64	500.0
longitude	(longitude)	float64	-500.0 500.0
spatial_ref	()	int32	4326

▼ Data variables:

LST_Day_1km	(time, latitude, longitude)	uint16	0 0 0 0 0 0 0 0 0...
LST_Night_1km	(time, latitude, longitude)	uint16	0 0 0 0 0 0 0 0 0...

▼ Attributes:

crs :	EPSG:4326
grid_mapping :	spatial_ref

In [12]:
```
combined = xr.merge([mod09, mod11], join='outer')
datacube_stack = Raster(combined, crs = 'EPSG:4326')
combined
```

Out[12]: xarray.Dataset

► Dimensions: (**time**: 192, **latitude**: 3, **longitude**: 4)

▼ Coordinates:

time	(time)	datetime64[ns]	2018-01-01 ... 2...
latitude	(latitude)	float64	-250.0 250.0 50...
longitude	(longitude)	float64	-500.0 -250.0 2...
spatial_ref	()	int32	4326

▼ Data variables:

sur_refl_b04	(time, latitude, longitude)	float64	nan -3.277e+04...
sur_refl_b07	(time, latitude, longitude)	float64	nan -3.277e+04...
LST_Day_1km	(time, latitude, longitude)	float64	nan nan nan na...
LST_Night_1km	(time, latitude, longitude)	float64	nan nan nan na...

▼ Attributes:

crs :	EPSG:4326
grid_mapping :	spatial_ref

In [13]:
```
s3 = boto3.resource('s3', config = Config(signature_version=UNSIGNED))
bucket = s3.Bucket("harvest-soc-features")
feat_urls = [f"https://s3.amazonaws.com/{bucket.name}/{obj.key}" for obj in bucket.
```

In [14]:
```
temp_files = []
for url in feat_urls:
    response = requests.get(url)
    temp = tempfile.NamedTemporaryFile(delete=False)
    temp.write(response.content)
    temp_files.append(temp.name)
```

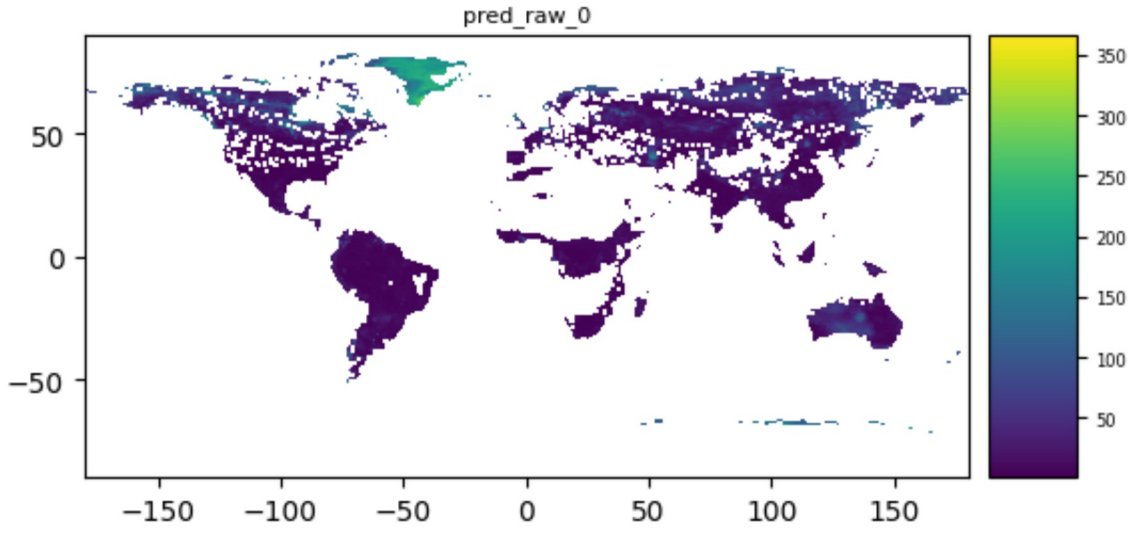
In [16]:
```
feature_stack = Raster(temp_files)
```

In [17]:
```
final_stack = feature_stack.append(datacube_stack)
```

In [20]:
```
training = rio.open('soctrainingdata.tif')
training_a = final_stack.extract_raster(training)
training_a = training_a.dropna()
X = training_a.drop(columns=["value", "geometry"]).values
y = training_a["value"].values
```

In [21]:
```
pipeline = Pipeline(
    [('scaling', StandardScaler()),
     ('regressor', RandomForestRegressor())]
)
pipeline.fit(X, y);
```

In [22]:
```
result = final_stack.predict(estimator=pipeline, dtype='float32') # predict based o
result.plot()
plt.show()
```



In [25]:
```
plt.savefig('output.png')
```

```
<Figure size 640x480 with 0 Axes>
```

In [ ]: