GPBay_Supervised_RS

January 12, 2023

[**Supervised Workflow**] Remote Sensing Analysis of Mangrove Forest Health and Extent in the Grand-Pierre Bay, Artibonite, Haiti

Written by Alexandre Erich Sebastien Georges, *PhD Student at UC Berkeley in EFMH-Civil and Environmental Engineering*, January 2023

[1]:
```
import os, pickle, itertools, glob, re, datetime

import cv2 as cv
import numpy as np
import xarray as xr
import pandas as pd
import geopandas as gpd
import rioxarray as rxr
import earthpy.plot as ep
import matplotlib.pyplot as plt


from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,␣
 ↪HistGradientBoostingClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score


import matplotlib.cm as cm
import matplotlib.patches as mpatches
from matplotlib import colors as colors_mat

import seaborn as sns

from tqdm.notebook import tqdm
```

1 Data Acquisition

1.1 AOI Selection

[2]:
```
aoi_list = ['shapes_Grand-Pierre', 'shapes_de la Tortue', 'shapes_Gonaives']
#aoi_list = ['north (horn)', 'northeast', 'west', 'center', 'east',␣
↪'southwest', 'south']
#aoi_list = ['coast_NW', 'coast_N', 'horn', 'coastline_gonave', 'marsh_E',␣
↪'marsh_W']
#aoi_list = ['west','east','barrier', 'coast_marsh_S']
```

1.2 PlanetLabs GeoTIFFs Acquisition

[3]:
```
# Pull paths of .shp files of AOIs
def aoi_path(name):
    return '../bay_shapes/'+name+'.shp'

def paths_to_datetimeindex(list):
    pattern = r'.*(\d{4}-\d{2}-\d{2}).*'
    new_list = []
    for item in list:
        time = re.search(pattern, item).group(1)
        time = datetime.datetime.strptime(time, '%Y-%m-%d').date()
        new_list.append(time)
    new_list = sorted(new_list)
    return new_list
```

[4]:
```
data_dir = 'E:/PhD Data/Planet 3m/*.TIF' # Make sure to have the right path for␣
↪data location

sites = []
resSites = []
#unmaskedSites = [] # Eliminated unmaskedSites as new workflow does not require␣
↪copying an unmasked version of TIFF files, all the data is coming from the␣
↪sites Dataset file.

band_names = {'band':{1: 'CoastalBlue',
                      2: 'Blue',
                      3: 'Green1',
                      4: 'Green',
                      5: 'Yellow',
                      6: 'Red',
                      7: 'RedEdge',
                      8: 'NIR'
                }}
```

```
band_titles = ['CoastalBlue', 'Blue', 'Green1', 'Green', 'Yellow', 'Red',␣
↪'RedEdge', 'NIR']
times = [i.strftime('%m/%d/%Y') for i in paths_to_datetimeindex(glob.
↪glob(data_dir))]
pos_var = xr.Variable('Position', ['Ob1','Ob2','Ob3','Ob4','Ob5','Ob6','Ob7'])
time_var = xr.Variable('Observation Date', times)


for area in aoi_list:
    aoi = gpd.read_file(aoi_path(area))
    geotiffs_da = xr.concat(
        [rxr.open_rasterio(entry).rio.clip(aoi.geometry, from_disk=True).
↪squeeze()
        for entry in glob.glob(data_dir)], dim=time_var)
    area_ds = geotiffs_da.to_dataset(dim='Observation Date')
    sites.append(area_ds)
    resSites.append(area_ds.rio.resolution()[0])
    #unmaskedSites.append(area_ds.copy())
```

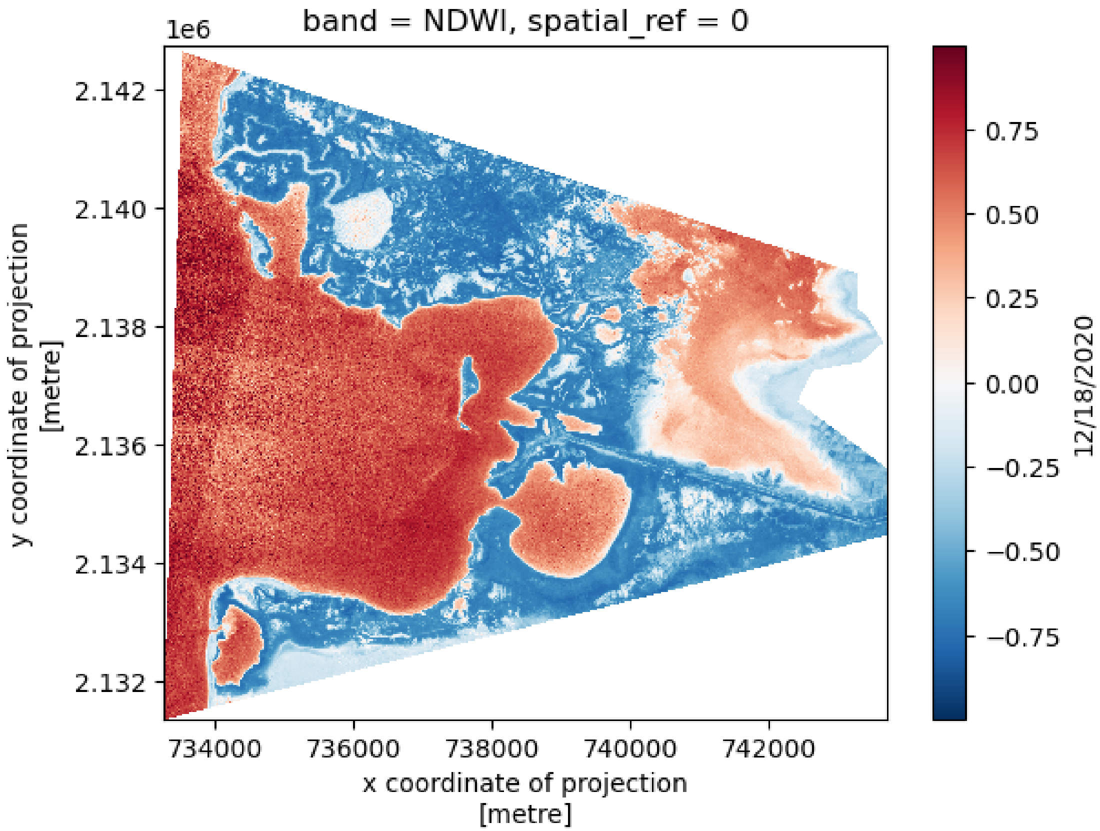
2 Band Math and Indexes

2.1 Normalized Difference Water Index (NDWI)

[5]:
```
# NDWI = (green - nir)/(green + nir) McFeeters (1996)

ndwiMasks= []
for i,site in enumerate(sites):
    ndwi_times = []
    for time in times:
        ndwi = (site[time][3] - site[time][7])/(site[time][3] + site[time][7])
        ndwi_times.append(xr.DataArray(ndwi, coords={'band' : 'NDWI'}))
        #site = xr.concat([site, ndwi_arr.to_dataset()], dim="band")
    ndwi_arr = (xr.concat(ndwi_times, dim=time_var)).
↪to_dataset(dim='Observation Date')
    ndwiMasks.append(ndwi_arr)
    sites[i] = xr.concat([site, ndwi_arr], dim="band")
```

[6]:
```
sites[0]['12/18/2020'][-1].plot()
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x267010ad3a0>
```

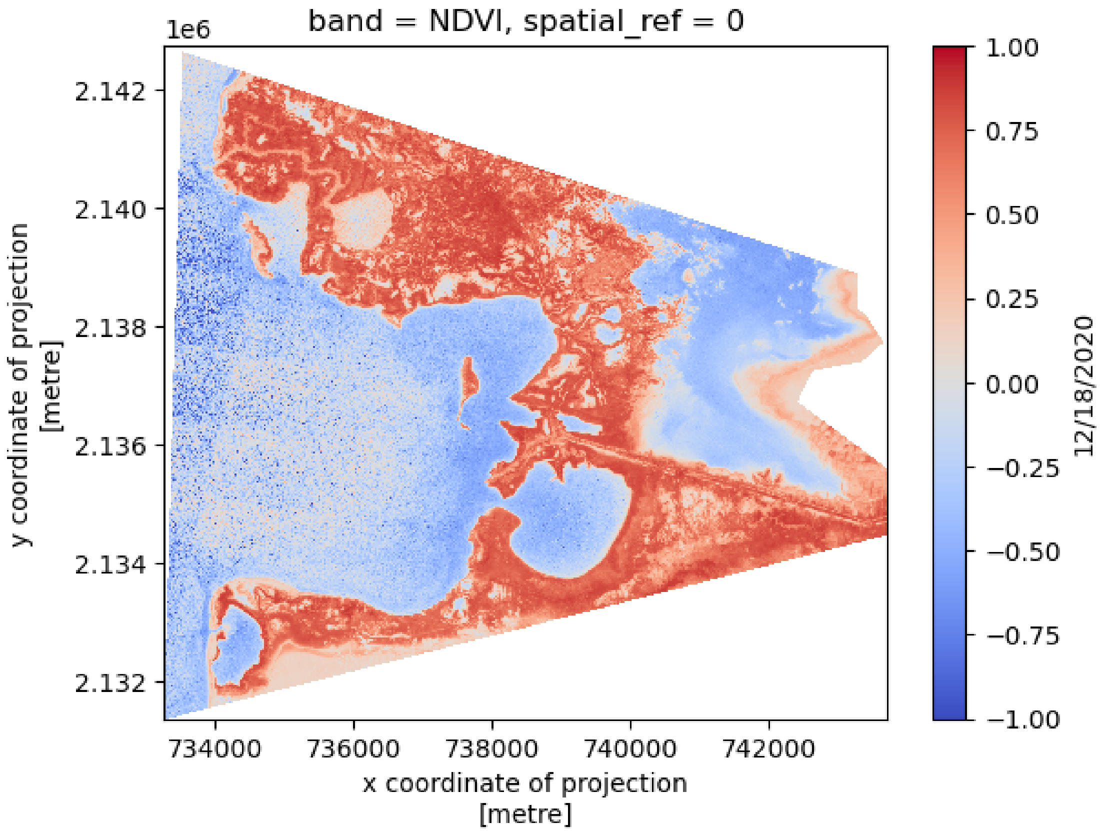


2.2 Normalized Difference Vegetation Index (NDVI)

[7]:
```
# NDVI = (nir - red)/(nir + red)

ndviSites = []
for i,site in enumerate(sites):
    ndvi_times = []
    for time in times:
        ndvi = (site[time][7] - site[time][5])/(site[time][7] + site[time][5])
        ndvi_times.append(xr.DataArray(ndvi, coords={'band' : 'NDVI'}))
        #site = xr.concat([site, ndwi_arr.to_dataset()], dim="band")
    ndvi_arr = (xr.concat(ndvi_times, dim=time_var)).
 ↪to_dataset(dim='Observation Date')
    ndviSites.append(ndvi_arr)
    sites[i] = xr.concat([site, ndvi_arr], dim="band")
```

[8]:
```
sites[0]['12/18/2020'][-1].plot(vmin=-1, vmax=1, cmap='coolwarm')
```

[8]: <matplotlib.collections.QuadMesh at 0x26701189130>

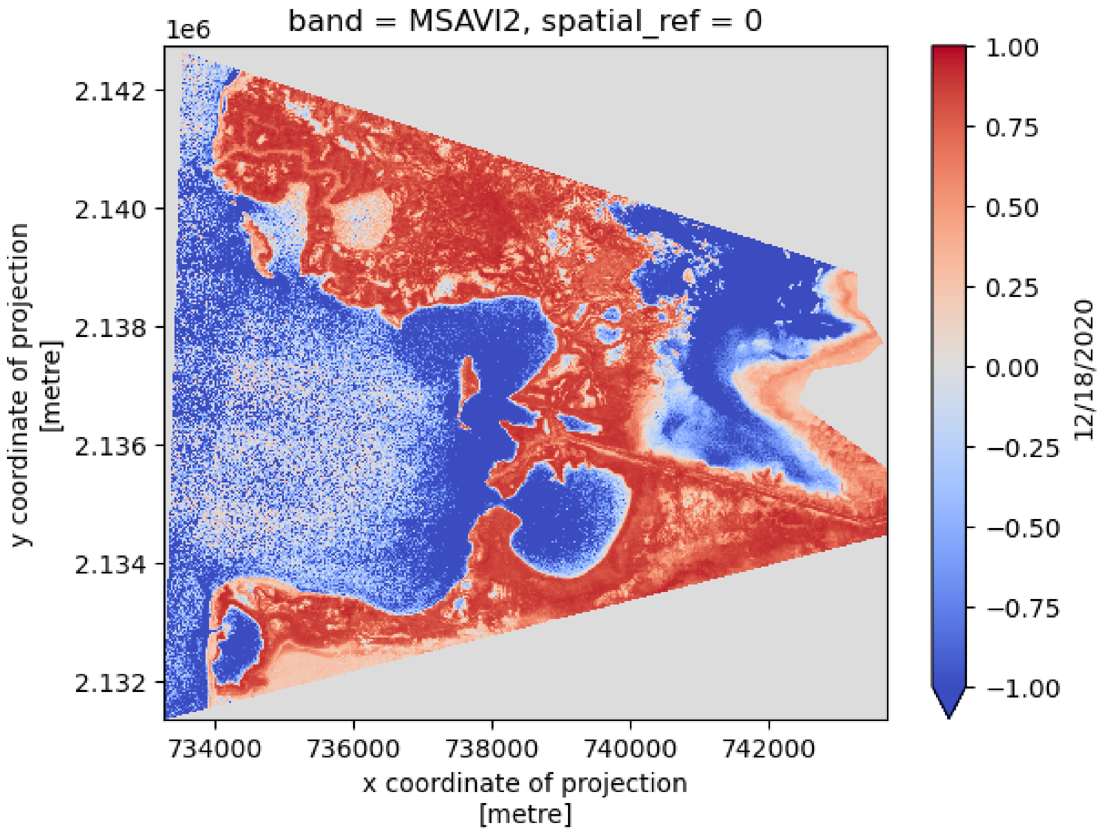


2.3 Modified Soil Adjusted Vegetation Index (MSAVI2)

[9]:
```
# msavi2 = (2 * nir + 1 - sqrt( (2 * nir + 1)^2 - 8 * (nir - red) )) / 2

for i,site in enumerate(sites):
    msavi_times = []
    for time in times:
        msavi = 0.5*(2*site[time][7] + 1 - np.sqrt(np.square(2*site[time][7]+1)␣
 ↪- 8*(site[time][7] - site[time][5])))
        msavi_times.append(xr.DataArray(msavi, coords={'band' : 'MSAVI2'}))
        #site = xr.concat([site, ndwi_arr.to_dataset()], dim="band")
    msavi_arr = (xr.concat(msavi_times, dim=time_var)).
 ↪to_dataset(dim='Observation Date')
    sites[i] = xr.concat([site, msavi_arr], dim="band")
```

[10]:
```
sites[0]['12/18/2020'][-1].plot(vmin=-1, vmax=1, cmap='coolwarm')
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x26701260f70>
```



3 Supervised Land Cover Classification

We will for now work exclusively with Random Forest for the sake of building this workflow. A separate .jupyter notebook will be dedicated to figure out the best algorithm to use and calibrating it. This will then be brough back and updated here as a final method.

Visualization Legend

[11]:
```
custom_cmap = colors_mat.ListedColormap(colors=['#1f78b4', '#488f52',␣
 ↪'#a6cee3', '#8e3b00', '#fdbf6f', '#f7fcf5', '#00ff11', '#8115d3'])
boundaries = [0, 1, 2, 3, 4, 5, 6, 7]
custom_norm = colors_mat.BoundaryNorm(boundaries, custom_cmap.N, clip=True)

patches = [mpatches.Patch(color='#1f78b4', label='Water'),
            mpatches.Patch(color='#488f52', label='Mangrove'),
            mpatches.Patch(color='#a6cee3', label='Salt Flat'),
            mpatches.Patch(color='#8e3b00', label='Mud Flat'),
            mpatches.Patch(color='#fdbf6f', label='Soil'),
            mpatches.Patch(color='#f7fcf5', label='Sand'),
            mpatches.Patch(color='#00ff11', label='Crops'),
```

```
            mpatches.Patch(color='#8115d3', label='Urban')]
```

3.1 Labelled Data Acquisition and Train/Test Split

[12]:
```
# Raster to be trained on
training_dir = 'E:/PhD Data/Planet 3m/2020-08-15.TIF'
training_obs = rxr.open_rasterio(training_dir).dropna(dim='band')
# Importing label points
training_set = gpd.read_file('../training_data/training_gp_mangrove_v2.shp')
# Dropping any possible NaNs from labels
training_set = training_set.dropna(axis=0, how='any')
```

[13]:
```
# Matching Label Points with Raster Points
training_set['samples'] = training_set.apply(lambda x: training_obs.rio.
 ↪clip([x['geometry']], from_disk=True).squeeze().values, axis=1)
training_set.head()
```

[13]:
```
    id                         geometry  \
0  0.0  POINT (735753.788 2146500.002)
1  0.0  POINT (743414.351 2144463.806)
2  0.0  POINT (736138.678 2139745.793)
3  1.0  POINT (734453.230 2140369.688)
4  1.0  POINT (743397.279 2147839.359)

                                             samples
0  [402.0, 344.0, 325.0, 87.0, 224.0, 29.0, 1.0, …
1  [333.0, 330.0, 502.0, 355.0, 474.0, 239.0, 198…
2  [261.0, 197.0, 308.0, 154.0, 306.0, 134.0, 98…
3  [317.0, 221.0, 439.0, 320.0, 463.0, 250.0, 582…
4  [355.0, 273.0, 551.0, 478.0, 608.0, 278.0, 768…
```

[14]:
```
# Reshape data
new = []
X_data = (training_set['samples']).apply(lambda x: x.reshape((1, -1)))
for arr in X_data.values:
    new.append(arr[0].tolist())
X_data = np.array(new)

# Split data in train and test
X_train, X_test, y_train, y_test = train_test_split(X_data, training_set['id'].
 ↪values, test_size=0.3, stratify= training_set['id'].values)
print(f"X_train Shape: {X_train.shape}\nX_test Shape: {X_test.shape} \ny_train␣
 ↪Shape: {y_train.shape}\ny_test Shape:{y_test.shape}")
```

```
X_train Shape: (120, 8)
X_test Shape: (52, 8)
y_train Shape: (120,)
y_test Shape:(52,)
```

3.2 Reshaping Data to be classified

[15]:
```
reshapedSites = []
for site in sites:
    shapes = tuple(site.dims[d] for d in ['band', 'y', 'x'])
    resh_times = []
    for time in times:
        acq = [band.values.reshape(((band.shape)[0])*((band.shape)[1]), 1) for␣
 ↪band in site[time]]
        resh = np.array(acq).reshape(shapes[0], shapes[1]*shapes[2]).transpose()
        resh_times.append(resh)
    reshapedSites.append(resh_times)
```

[16]:
```
hgb = HistGradientBoostingClassifier()
hgb.fit(X_train, y_train)
```

[16]:
```
HistGradientBoostingClassifier()
```

[17]:
```
classifiedSites = []
for resh in tqdm(reshapedSites):
    time_pred = []
    for obs in tqdm(resh):
        # Filling NaNs with -999
        #obs = np.where(np.isnan(obs), np.ma.array(obs,
         #      mask = -999))
        time_pred.append(hgb.predict(obs[:,:8]))
    classifiedSites.append(time_pred)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]

  0%|          | 0/5 [00:00<?, ?it/s]

  0%|          | 0/5 [00:00<?, ?it/s]

  0%|          | 0/5 [00:00<?, ?it/s]
```

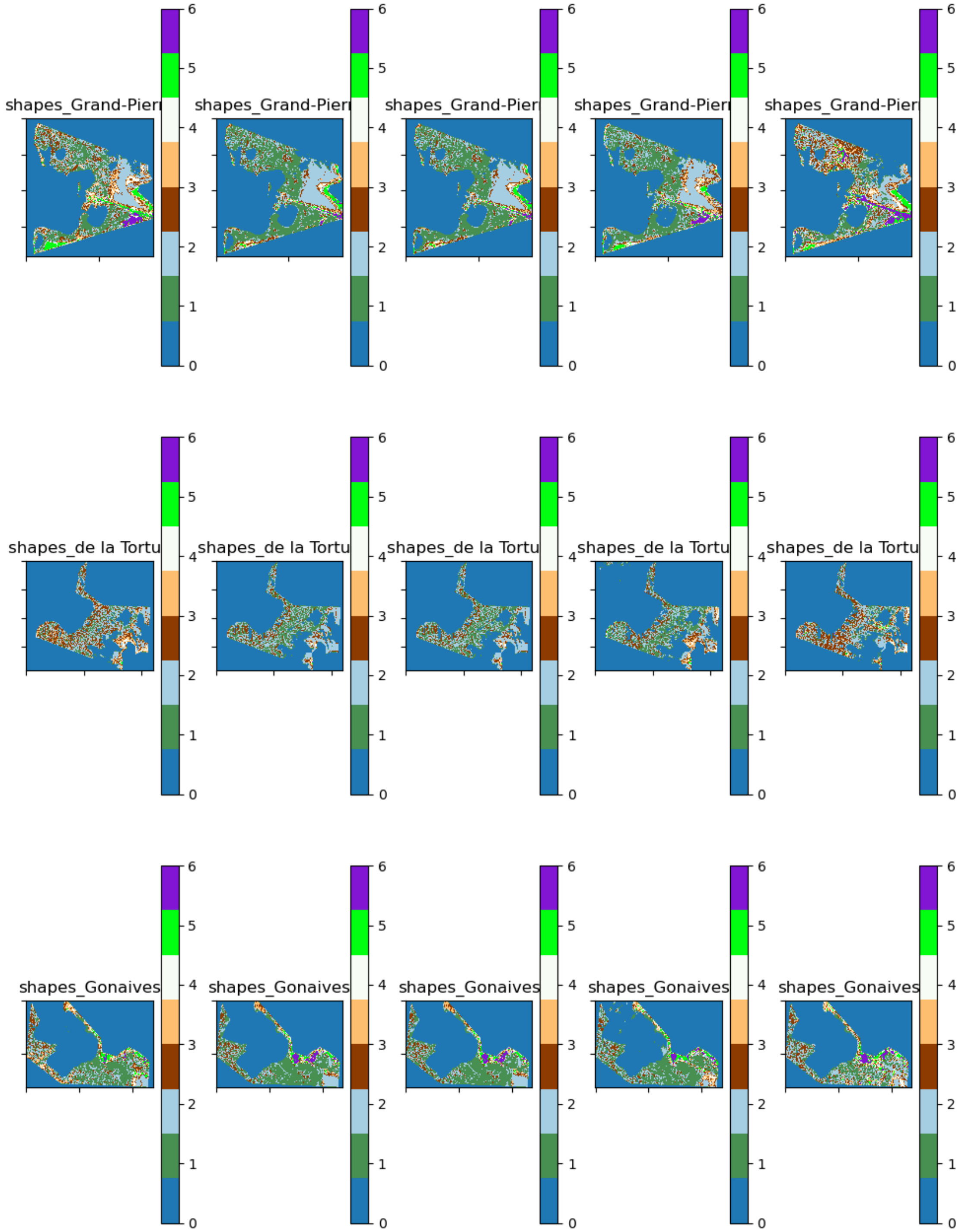
4 Post-Processing for Analysis

4.1 CannyEdge and Gaussian Blur for Visualization

[18]:
```
classifiedSites_r = []
classifiedSites_blur = []
for i, site in enumerate(classifiedSites):
    sites_r = []
    sites_blur = []
    for j, time in enumerate(times):
        obs_r = site[j].reshape(sites[i][time][0].shape).astype("uint8")
        obs_blur = cv.GaussianBlur(obs_r, (3,3), 0)
        sites_r.append(obs_r)
```

```
        sites_blur.append(obs_blur)
    classifiedSites_r.append(sites_r)
    classifiedSites_blur.append(sites_blur)
```

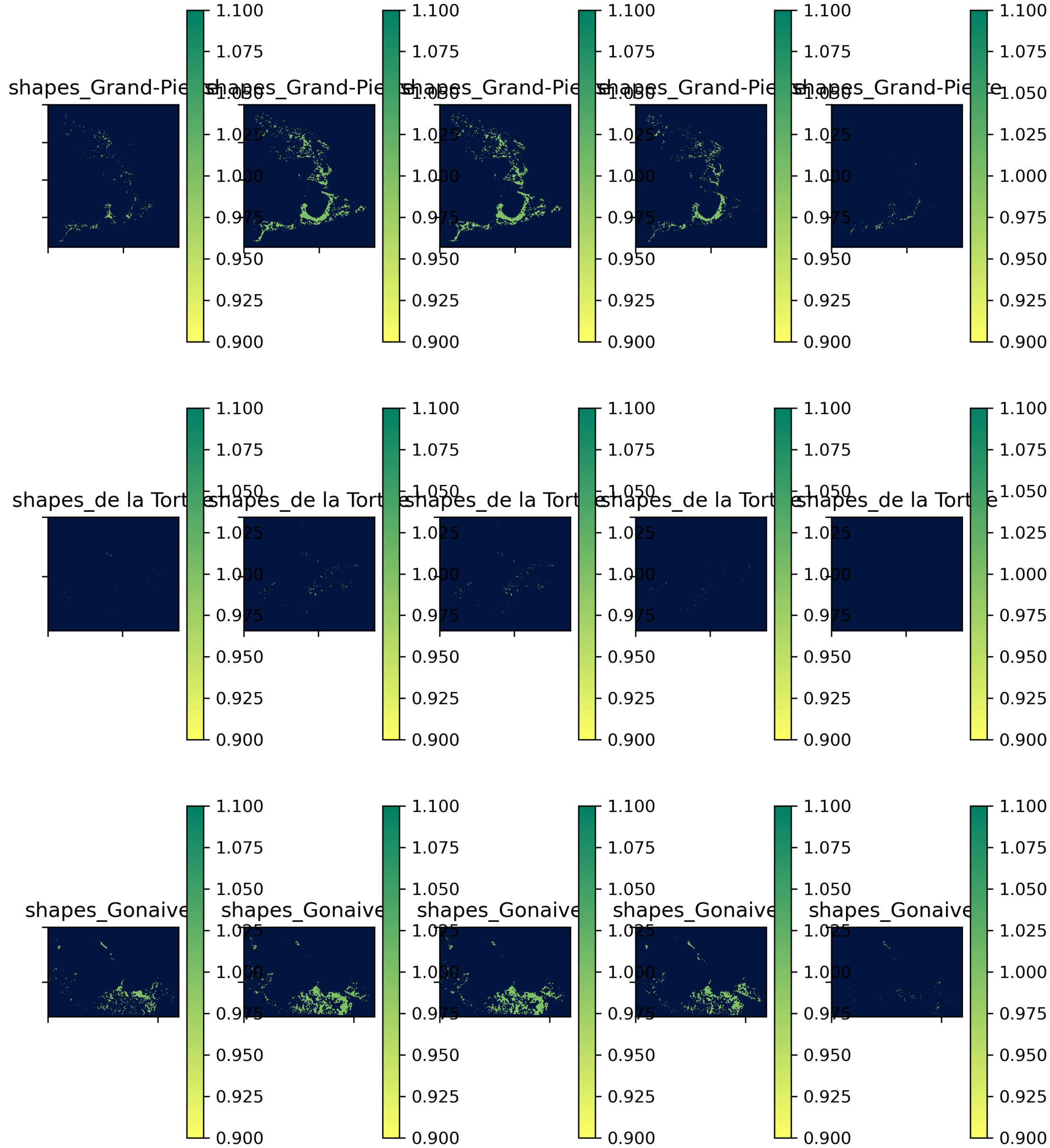
[19]:
```
fig, axs = plt.subplots(3,5, figsize=(12,16))
axs = list(itertools.chain.from_iterable(axs))
count = 0
for j,blur in enumerate(classifiedSites_blur):
    for ob in blur:
        nd = axs[count].imshow(ob, cmap=custom_cmap)
        axs[count].set_xticklabels([])
        axs[count].set_yticklabels([])
        axs[count].set_title(aoi_list[j])
        plt.colorbar(nd, ax = axs[count])
        count+=1
plt.show()
```


shapes_Grand-Pierr
shapes_Grand-Pierr
shapes_Grand-Pierr
shapes_Grand-Pierr
shapes_Grand-Pierr
shapes_de la Tortu
shapes_de la Tortu
shapes_de la Tortu
shapes_de la Tortu
shapes_de la Tortu
shapes_Gonaives
shapes_Gonaives
shapes_Gonaives
shapes_Gonaives
shapes_Gonaives
0 1 2 3 4 5 6

4.2 Masking and Keeping Mangrove Cover

[20]:
```
mangroveSites = []
mangroveSites_blur = []
for i, site in enumerate(classifiedSites_blur):
    # Masking everything else and keeping mangrove cover
    masks_blur = [np.ma.masked_where(bl != 1, bl) for bl in site]
    masks = [np.ma.masked_where(pred != 1, pred) for pred in classifiedSites_r]
    mangroveSites_blur.append(masks_blur)
    mangroveSites.append(masks_blur)
```

[21]:
```
fig, axs = plt.subplots(3,5, figsize=(10,12), dpi=300)
axs = list(itertools.chain.from_iterable(axs))
count = 0
for j,blur in enumerate(mangroveSites):
    for ob in blur:
        axs[count].set_facecolor('xkcd:navy')
        nd = axs[count].imshow(ob, cmap='summer_r')
        axs[count].set_xticklabels([])
        axs[count].set_yticklabels([])
        axs[count].set_title(aoi_list[j])
        plt.colorbar(nd, ax = axs[count])
        count+=1
plt.show()
```



5 Analysis

5.1 Gross Cover and 2MOA Calculations

[22]:
```
def moa_calc(inputSites, dir): # Inputs are region to calculate MOA on and the
    ↪direction to take MOA on, dir = 'x' or 'y'
    secMOAs = []
    for j in range(5):
        moa_yr = []
```

```
        for i, input in enumerate(inputSites):
            # X direction MOA
            if dir == 0:
                pixs = [row.count() for row in input[j]]
                dists = [3*((input[j].shape[0])/2 - 1 - k) for k in␣
↪range(len(input[j]))]
            # Y direction MOA
            elif dir == 1:
                pixs = [col.count() for col in input[j].T]
                dists = [3*(((input[j].T).shape[0])/2 - 1 - k) for k in␣
↪range(len(input[j].T))]

            moa = 0
            for l,pix in enumerate(pixs):
                moa += (3**2)*pix * dists[l]**2
            moa_yr.append(moa)
        secMOAs.append(moa_yr)

    changeMOA = [secMOAs[-1][i] - secMOAs[0][i] for i in range(len(secMOAs[0]))]
    percMOA = [(change*100)/(secMOAs[0][i]) for i, change in␣
↪enumerate(changeMOA)]

    return secMOAs, changeMOA, percMOA
```

[23]:
```
def get_metrics(inputSites):
    pixelSites = []
    areaSites = []
    changeSites = []
    percChangeSites = []
    secMOAs_x = []

    # Counting Pixels for Area calculations
    for i, mask in enumerate(inputSites):
        pixels = np.array([np.size(m.mask) - np.count_nonzero(m.mask) for m in␣
↪mask])
        pixelSites.append(pixels)

    # Calcualte Gross Cover Change
    for i, pixels in enumerate(pixelSites):
        surfaces = [(3**2)*pixel_obs for pixel_obs in pixels]
        areaSites.append(surfaces)
        change = (surfaces[-1] - surfaces[0])/(1e6)
        percent = (change*100)/(surfaces[0]/1e6)
        changeSites.append(change)
        percChangeSites.append(percent)

    # Calculate MOA in x and y
```

```
    secMOAs_x, changeMOA_x, percMOA_x = moa_calc(inputSites, 0)
    secMOAs_y, changeMOA_y, percMOA_y = moa_calc(inputSites, 1)

    # Calculate Polar MOA
    secMOA_pol = []
    for i, year in enumerate(secMOAs_x):
            moa_yr = [moa_x + secMOAs_y[i][j] for j, moa_x in enumerate(year)]
            secMOA_pol.append(moa_yr)

    changeMOA_pol = [secMOA_pol[-1][i] - secMOA_pol[0][i] for i in␣
 ↪range(len(secMOA_pol[0]))]
    percMOA_pol = [(change*100)/(secMOA_pol[0][i]) for i, change in␣
 ↪enumerate(changeMOA_pol)]

    return areaSites, changeSites, percChangeSites, secMOAs_x, changeMOA_x,␣
 ↪percMOA_x, secMOAs_y, changeMOA_y, percMOA_y, secMOA_pol, changeMOA_pol,␣
 ↪percMOA_pol
```

[24]:
```
# areaSites, changeSites, percChangeSites, secMOAs_x, changeMOA_x, percMOA_x,␣
 ↪secMOAs_y, changeMOA_y, percMOA_y, secMOA_pol, changeMOA_pol, percMOA_pol
metrics_healthy = get_metrics(mangroveSites)

df_mangrove_analysis = pd.DataFrame({'AOI': aoi_list, 'Gross Cover Change␣
 ↪(km^2)': metrics_healthy[1],
                'Gross Percent Change (%)': metrics_healthy[2],
                'Polar 2MOA Change (2022-2012)': metrics_healthy[-2],
                'Polar 2MOA Percent Change (%)': metrics_healthy[-1]})
df_mangrove_analysis
```

[24]:

	AOI	Gross Cover Change (km^2)	Gross Percent Change (%)
0	shapes_Grand-Pierre	-4.507479	-28.770890
1	shapes_de la Tortue	-3.402000	-31.190538
2	shapes_Gonaives	-1.287972	-23.853281

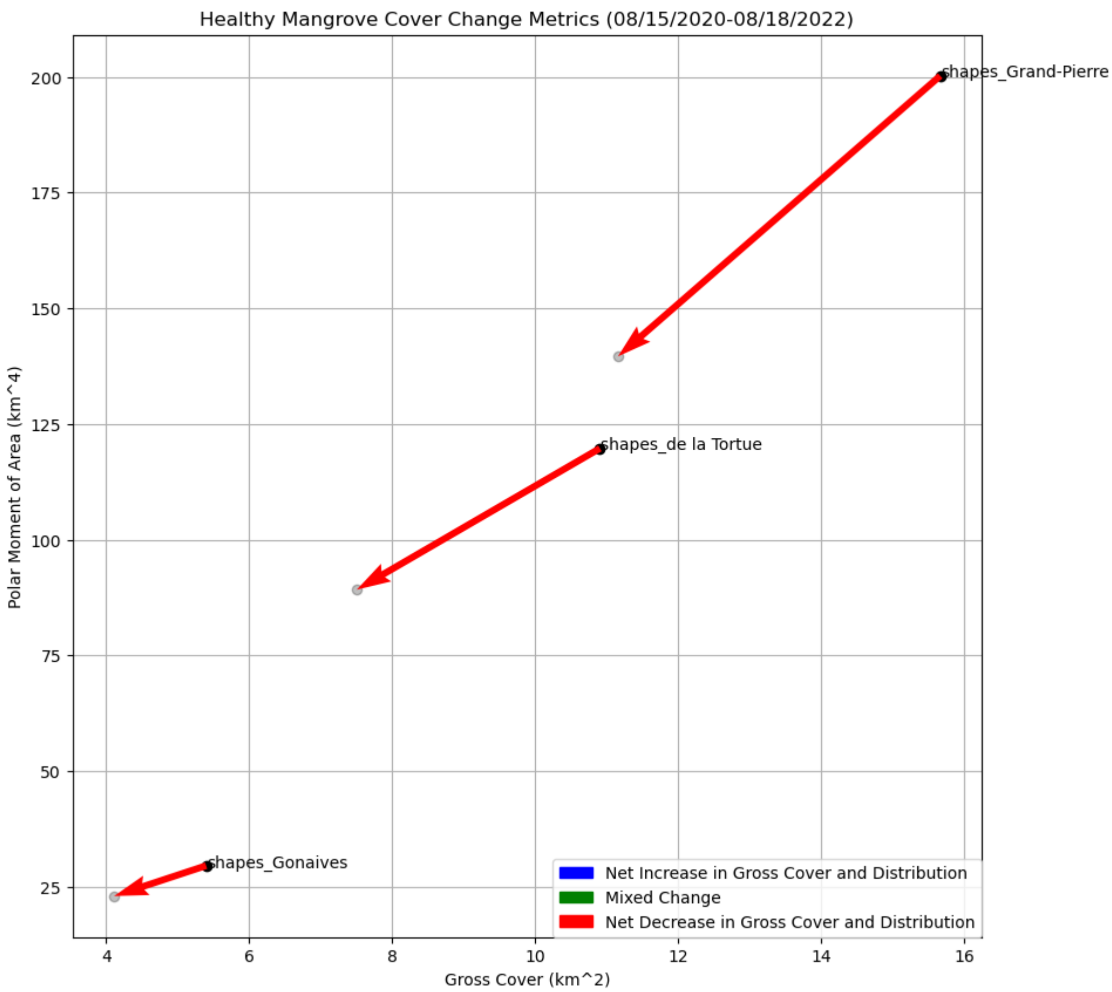
	Polar 2MOA Change (2022-2012)	Polar 2MOA Percent Change (%)
0	-6.061432e+13	-30.276772
1	-3.053151e+13	-25.500635
2	-6.568127e+12	-22.202787

5.2 Mangrove Extent Metrics

[25]:
```
# areaSites, changeSites, percChangeSites, secMOAs_x, changeMOA_x, percMOA_x,␣
 ↪secMOAs_y, changeMOA_y, percMOA_y, secMOA_pol, changeMOA_pol, percMOA_pol
def metrics_visualization(metric, title):
    fig, ax = plt.subplots(figsize=(10,10))
    ax.set_xlabel('Gross Cover (km^2)')
    ax.set_ylabel('Polar Moment of Area (km^4)')
```

```
    # Start positions (X is set of gross covers at Y1, Y is set of 2MOA_pol at␣
↪Y1)
    areaSites = np.array(metric[0])
    X = (areaSites.T[0])/1e6
    Y = np.array((metric[-3])[0])/1e12
    # Magnitude is distance change (U is for gross cover change, V is for 2MOA␣
↪change)
    U = metric[1]
    V = np.array(metric[-2])/1e12
    # Color code of arrows (blue for increase in both, red for decrease in␣
↪both, gray for other)
    arr_colors = []
    for i, change in enumerate(metric[1]):
        if change > 0 and (metric[-2])[i] > 0:
            arr_colors.append('blue')
        elif change < 0 and (metric[-2])[i] < 0:
            arr_colors.append('red')
        else:
            arr_colors.append('green')
    ax.grid()
    ax.scatter(x=X, y=Y, c='k')
    ax.scatter(x=areaSites.T[-1]/1e6, y=np.array((metric[-3])[-1])/1e12, c='k',␣
↪alpha=0.25)
    ax.quiver(X,Y,U,V, color=arr_colors, angles='xy', scale_units='xy', scale=1)
    for i, label in enumerate(aoi_list):
        plt.annotate(label, (X[i] + 0.005, Y[i] - 0.25,))

    patches = [mpatches.Patch(color='blue', label='Net Increase in Gross Cover␣
↪and Distribution'),
               mpatches.Patch(color='green', label='Mixed Change'),
               mpatches.Patch(color='red', label='Net Decrease in Gross Cover␣
↪and Distribution')]
    ax.legend(handles=patches, loc='lower right', borderaxespad=0.)
    ax.set_title(title)
    ax.patch.set_facecolor('xkcd:white')
    ax.set_axisbelow(True)
```

[26]:
```
metrics_visualization(metrics_healthy, 'Healthy Mangrove Cover Change Metrics␣
↪('+times[0]+'-'+times[-1]+')')
```



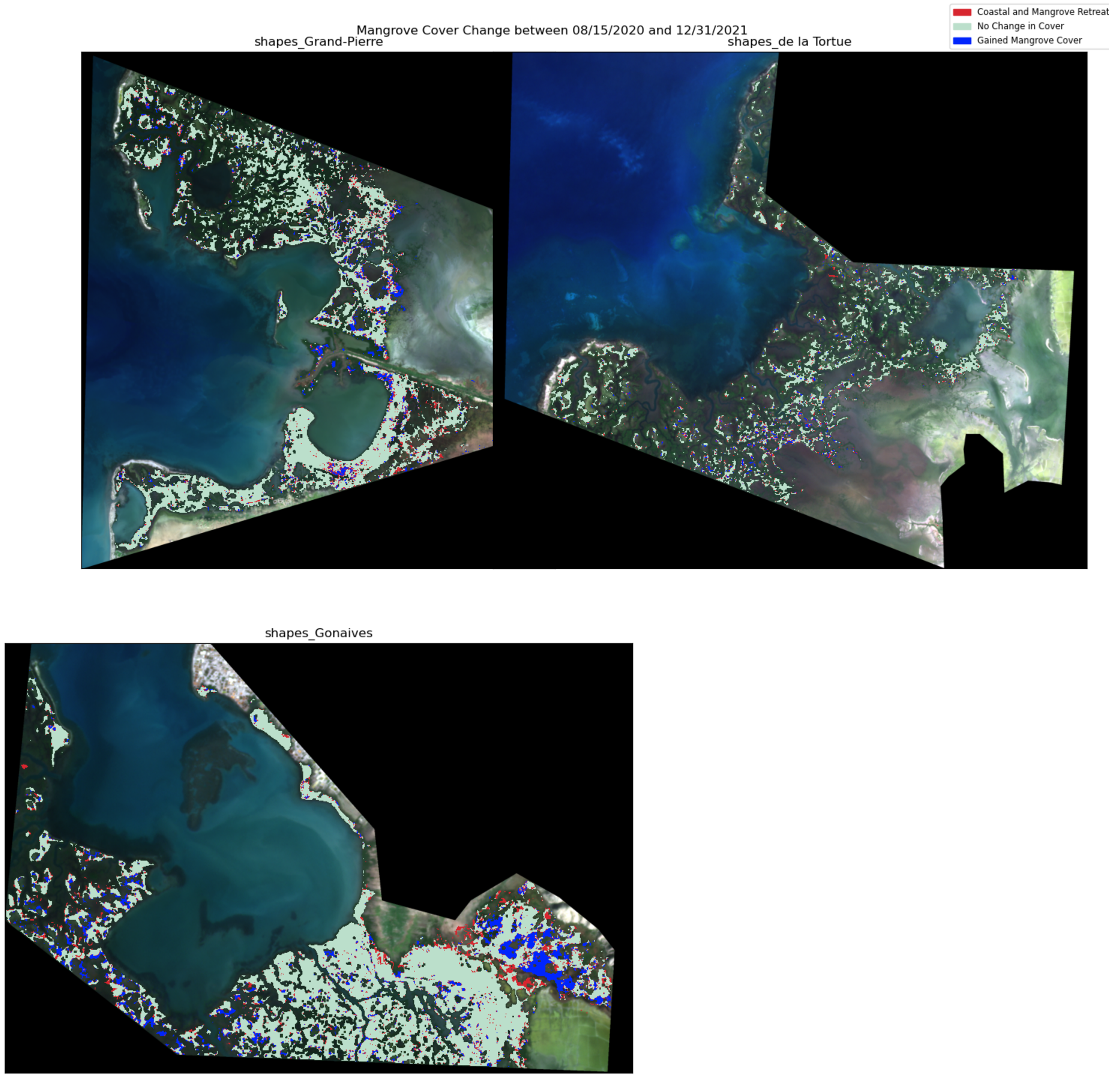
5.3 Mangrove Cover Change Visualizations

[27]:
```
np.unique(mangroveSites[0][0])
```

[27]:
```
masked_array(data=[1, --],
             mask=[False,  True],
       fill_value=999999,
            dtype=uint8)
```

[28]:
```
fig, axs = plt.subplots(2,2, figsize=(15,15))
axs = list(itertools.chain.from_iterable(axs))
custom_cmap = colors_mat.ListedColormap(['#0025FC', '#D7262F', '#BBDFCB'])
boundaries = [0, 2, 4, 6]
custom_norm = colors_mat.BoundaryNorm(boundaries, custom_cmap.N, clip=True)
#colors = [cm.get_cmap(value) for value in custom_cmap]
```

```
patches = [mpatches.Patch(color='#D7262F', label='Coastal and Mangrove␣
↪Retreat'),
           mpatches.Patch(color='#BBDFCB', label='No Change in Cover'),
           mpatches.Patch(color='#0025FC', label='Gained Mangrove Cover')]

for i,site in enumerate(mangroveSites_blur):
    ep.plot_rgb(sites[i][times[-1]], rgb=[5,3,1], stretch=True,
            str_clip=.09, ax=axs[i])
    # Apply the blurred Image mask to NDVI !!!!!
    first = site[0]
    last = site[-1]
    first = np.where(site[0] == 1, 4, 0)
    last = np.where(site[-2] == 1, 2, 0)
    change = first + last
    change_mask = np.ma.masked_where(change == 0, change)
    axs[i].imshow(change_mask, cmap=custom_cmap)
    axs[i].set_title(aoi_list[i])

fig.legend(handles=patches, fontsize='small')
fig.suptitle('Mangrove Cover Change between '+times[0]+' and '+times[-2])
fig.patch.set_facecolor('xkcd:white')
fig.delaxes(axs[-1])
#fig.delaxes(axs[-2])
plt.tight_layout(h_pad=5, w_pad=-15)
plt.show()
```



5.4 UVVR Calculations

The unvegetated-vegetated marsh ratio (UVVR) is a spatially integrative metric that correlates with sediment budgets and sea-level rise *(Ganju et al. 2017)*; it is defined as:

$$UVVR = \frac{A_{uv}}{A_v}$$

where Auv is the unvegetated area within a specified domain, and Av is the vegetated area. The total area of the wetland domain, Ad, is the sum of Auv and Av and the vegetated fraction, Fv, is therefore:

$$F_v = \frac{A_v}{A_d}$$

Unvegetated areas can represent bare sediment, pools, channels, and intertidal flats. Vegetated areas are typically wetland plain areas, and in a "binary" context, any vegetated plain, regardless of stem density, would be considered vegetated at some nominal spatial scale.

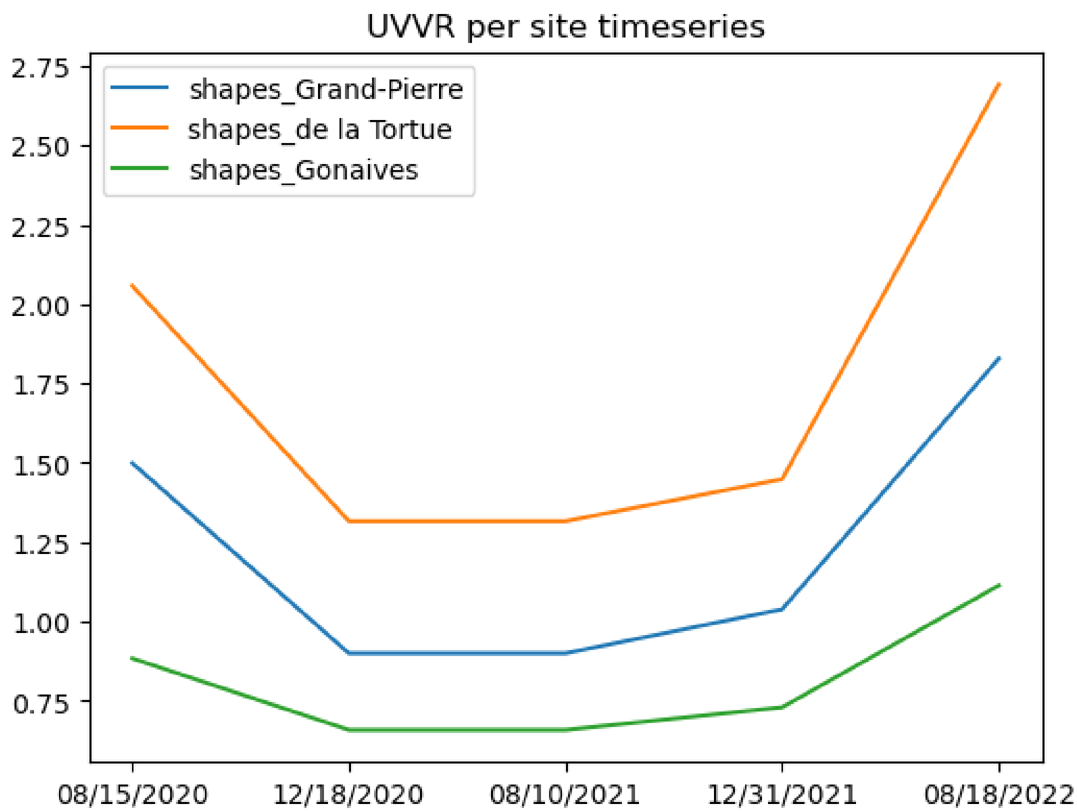
[29]:
```
vegSites = mangroveSites
unvegSites = []

# Using same masking procedure to get unvegetated areas
for site in classifiedSites_r:
    # Masking everything else and keeping mangrove cover
    masks = [np.ma.masked_where((pred != 2) & (pred != 3) & (pred != 4) & (pred␣
 ↪!= 5), pred) for pred in site]
    unvegSites.append(masks)
```

[30]:
```
# areaSites, changeSites, percChangeSites, secMOAs_x, changeMOA_x, percMOA_x,␣
 ↪secMOAs_y, changeMOA_y, percMOA_y, secMOA_pol, changeMOA_pol, percMOA_pol

unvegArea = get_metrics(unvegSites)[0]
vegArea = get_metrics(vegSites)[0]
uvvrSites = []
for i,site in enumerate(unvegArea):
    uvvr = [np.array(site) / np.array(vegArea[i])]
    uvvrSites.append(uvvr)
```

[31]:
```
df_uvvr = pd.DataFrame({'AOI': aoi_list})
for i,time in enumerate(times):
    df_uvvr[time] = (np.array(uvvrSites).transpose()[i][0])
df_uvvr
```

[31]:
```
                  AOI  08/15/2020  12/18/2020  08/10/2021  12/31/2021  \
0  shapes_Grand-Pierre    1.499326    0.900048    0.900048    1.038571
1  shapes_de la Tortue    2.057637    1.316253    1.316253    1.448450
2      shapes_Gonaives    0.884186    0.658744    0.658744    0.729837

   08/18/2022
0    1.828353
1    2.692501
2    1.113566
```

[32]:
```
fig, ax = plt.subplots()
for i, aoi in enumerate(aoi_list):
    ax.plot(df_uvvr.iloc[i][1:], label=aoi)
ax.set_title('UVVR per site timeseries')
ax.legend()

plt.show()
```



5.5 Mangrove Health Indices Timeseries

5.5.1 Water Masking

In order to get accurate trends for vegetation alone, we are masking out surface water using our calculated NDWIs.

[33]:
```
# Reminder that we calculated ndwi_masks earlier
ndwiMasks
```

[33]:
```
[<xarray.Dataset>
 Dimensions:     (y: 3795, x: 3482)
 Coordinates:
     band        <U4 'NDWI'
 Dimensions without coordinates: y, x
 Data variables:
     08/15/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
     12/18/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
     08/10/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
     12/31/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
     08/18/2022  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan,
 <xarray.Dataset>
```

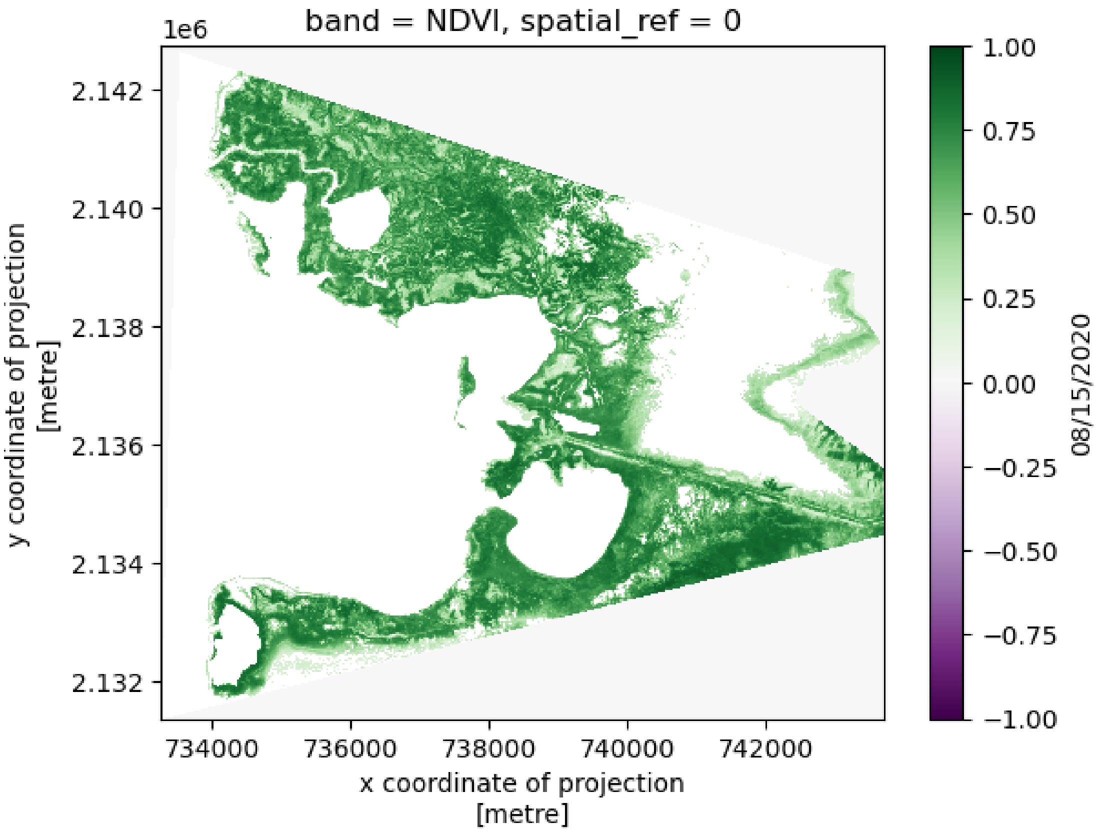
```
Dimensions:     (y: 3820, x: 4391)
Coordinates:
    band        <U4 'NDWI'
Dimensions without coordinates: y, x
Data variables:
    08/15/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    12/18/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    08/10/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    12/31/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    08/18/2022  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan,
<xarray.Dataset>
Dimensions:     (y: 1629, x: 2377)
Coordinates:
    band        <U4 'NDWI'
Dimensions without coordinates: y, x
Data variables:
    08/15/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    12/18/2020  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    08/10/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    12/31/2021  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan
    08/18/2022  (y, x) float32 nan nan nan nan nan nan … nan nan nan nan nan]
```

[34]:
```
# Masking water out of observations
mask_count = 0
for i, site in enumerate(sites):
    for time in times:
        # Masking out water in NDVI band
        new_ndvi = np.ma.masked_where(ndwiMasks[i][time] > -0.2, site[time][-2])
        # Masking out water in MSAVI2 band
        new_msavi = np.ma.masked_where(ndwiMasks[i][time] > -0.2,␣
 ↪site[time][-1])
        # Dropping out NaNs
        site[time][-1] = np.nan_to_num(new_msavi, nan=0)
        site[time][-2] = np.nan_to_num(new_ndvi, nan=0)
        mask_count+=1
```

5.5.2 Average NDVI timeseries

[35]:
```
sites[0][times[0]][-2].plot(vmin=-1, vmax=1, cmap='PRGn')
```

[35]:
```
<matplotlib.collections.QuadMesh at 0x26857b6a070>
```

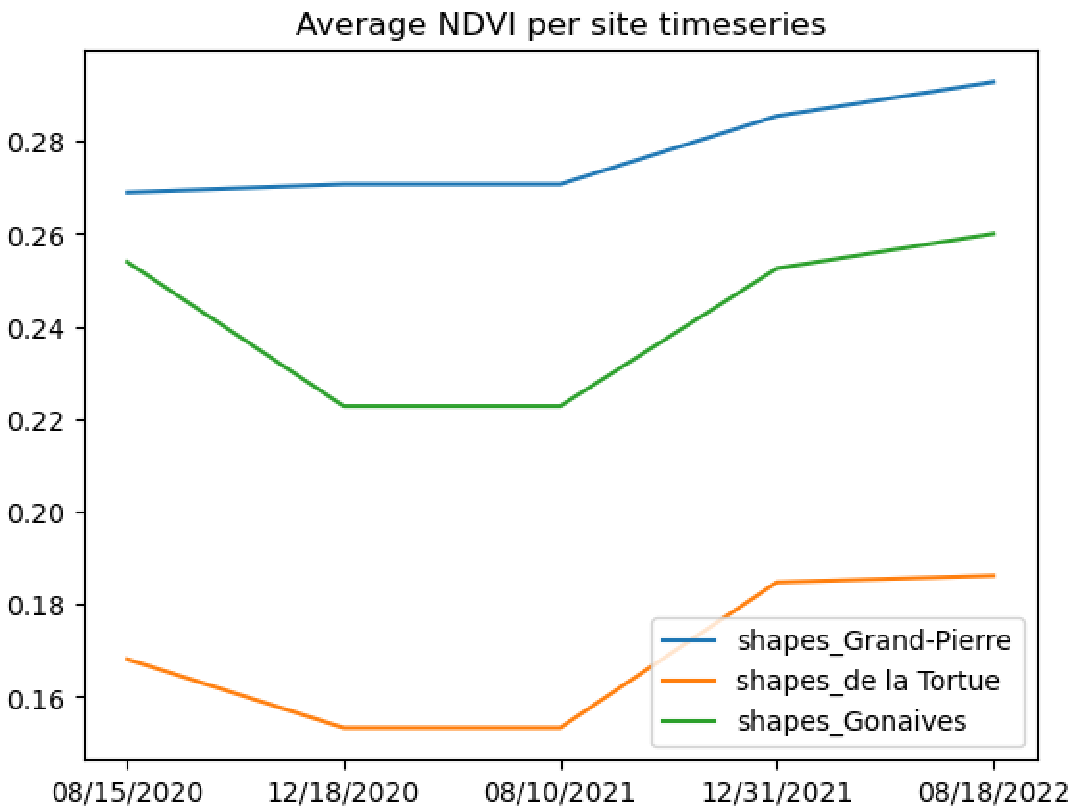


[36]:
```
avg_NDVI = []
for site in sites:
    avg = [site[time][-2].mean().values for time in times]
    avg_NDVI.append(avg)
```

[37]:
```
df_ndvi = pd.DataFrame({'AOI': aoi_list})
for i,time in enumerate(times):
    df_ndvi[time] = (np.array(avg_NDVI).transpose()[i])
df_ndvi
```

[37]:
```
                    AOI  08/15/2020  12/18/2020  08/10/2021  12/31/2021  \
0  shapes_Grand-Pierre    0.268952    0.270696    0.270696    0.285382
1  shapes_de la Tortue    0.168117    0.153325    0.153325    0.184721
2       shapes_Gonaives    0.253935    0.222767    0.222767    0.252499

   08/18/2022
0    0.292732
1    0.186136
2    0.259977
```

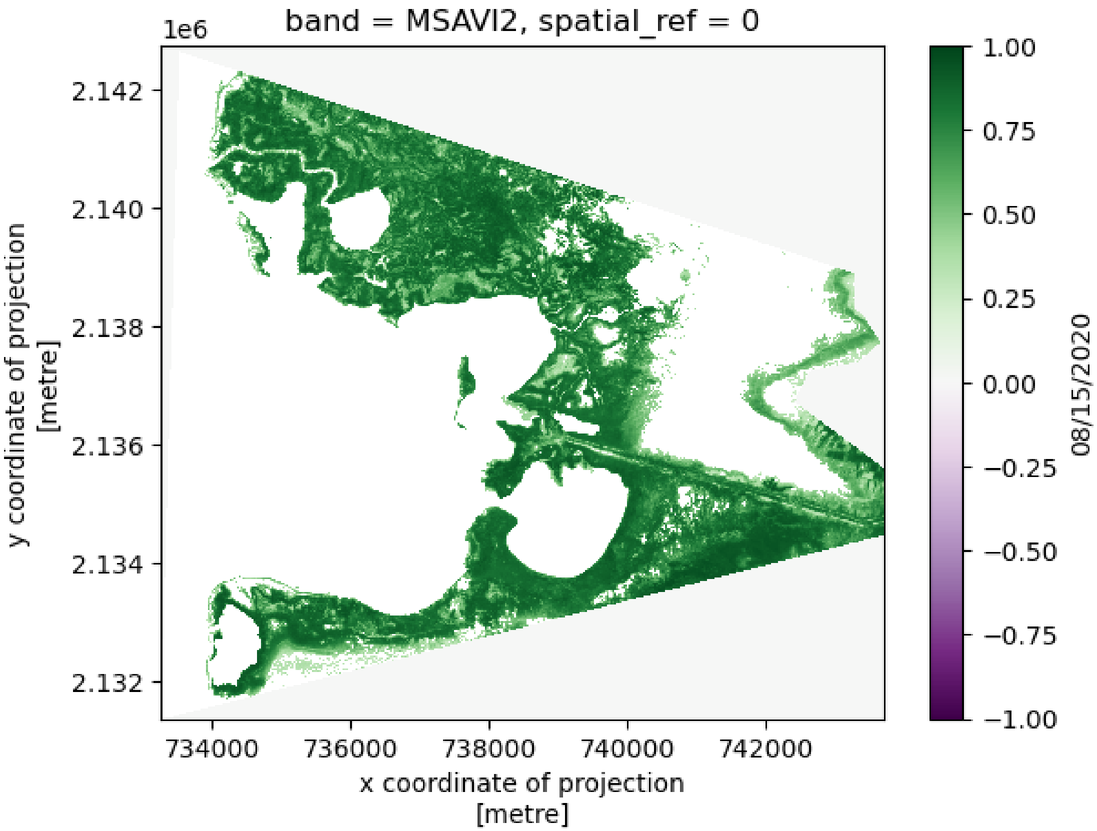
```
[38]: fig, ax = plt.subplots()
      for i, aoi in enumerate(aoi_list):
          ax.plot(df_ndvi.iloc[i][1:], label=aoi)
      ax.set_title('Average NDVI per site timeseries')
      ax.legend()

      plt.show()
```



5.5.3 Average MSAVI2 timeseries

```
[39]: sites[0][times[0]][-1].plot(vmin=-1, vmax=1, cmap='PRGn')
```

```
[39]: <matplotlib.collections.QuadMesh at 0x26798a04a30>
```

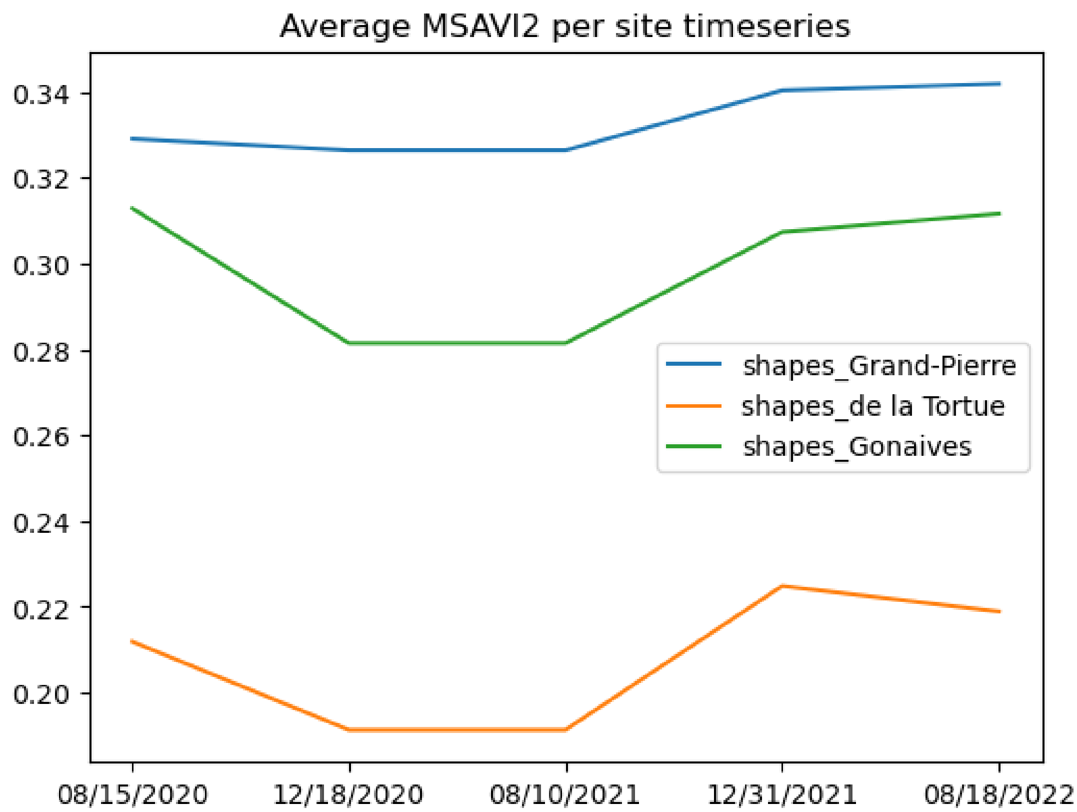


[40]:
```
avg_MSAVI2 = []
for site in sites:
    avg = [site[time][-1].mean().values for time in times]
    avg_MSAVI2.append(avg)
```

[41]:
```
df_msavi = pd.DataFrame({'AOI': aoi_list})
for i,time in enumerate(times):
    df_msavi[time] = (np.array(avg_MSAVI2).transpose()[i])
df_msavi
```

[41]:
```
                    AOI  08/15/2020  12/18/2020  08/10/2021  12/31/2021  \
0  shapes_Grand-Pierre    0.329059    0.326368    0.326368    0.340335
1  shapes_de la Tortue    0.211840    0.191272    0.191272    0.224815
2      shapes_Gonaives    0.312847    0.281408    0.281408    0.307339

   08/18/2022
0    0.341868
1    0.218883
2    0.311590
```

[42]:
```
fig, ax = plt.subplots()
for i, aoi in enumerate(aoi_list):
    ax.plot(df_msavi.iloc[i][1:], label=aoi)
ax.set_title('Average MSAVI2 per site timeseries')
ax.legend()

plt.show()
```



5.6 dNVDI Mangrove Health Analysis

[43]:
```
dndvi = [site[times[-1]][-2] - site[times[0]][-2] for site in sites]

fig, axs = plt.subplots(3,1, figsize=(12,12))

fig.suptitle('Three Bays Mangrove Forest\ndNDVI ['+times[0]+'-'+times[-1]+']')
#axs = list(itertools.chain.from_iterable(axs))
cmaps = ['Set1', 'Pastel2']
divnorm = colors_mat.TwoSlopeNorm(vmin=-1, vmax=1, vcenter=0)

for i,diff in enumerate(dndvi):
    #colors = ["darkorange", "black", "lawngreen"]
```

```
    #cmap1 = LinearSegmentedColormap.from_list('viridis', colors)
    nd = axs[i].imshow(diff, cmap='seismic_r', norm=divnorm)
    axs[i].set_xticklabels([])
    axs[i].set_yticklabels([])
    axs[i].set_title(aoi_list[i]+'\ndNDVI')
    plt.colorbar(nd, ax=axs[i])
plt.tight_layout(h_pad=0, w_pad=-35)
#fig.delaxes(axs[-1])
plt.show()
```

Three Bays Mangrove Forest
dNDVI [08/15/2020-08/18/2022]