APPENDIX B

Chapter 3 Training Module Overview

These training modules are provided as Python scripts in the form of Jupyter Notebooks (http://jupyter.org). The appeal of using Jupyter Notebooks is that Jupyter Notebook servers can be deployed from any platform/operating system and natively deploys a range of open source programming and scripting languages that are supported by a very large, growing open source developer community. Jupyter notebooks support the Python and R languages, amongst others. This means that all open source packages supported by Python (e.g., GDAL) are readily available in Jupyter. Also, Jupyter Notebooks are cloud-friendly, as servers can be launched on high-performance cloud instances and displayed via any web broweser. Terminals are also supported. The developed Notebooks for the SERVIR training courses have instructions for setting up and using Notebooks. A suite of Notebooks has been developed to cover various aspects of SAR data processing and analysis with a focus on forest mapping. To exercise the Notebooks, several example SAR time series data stacks have been provided for:

- West Africa Region - Sub-Sahelian Forest and Savanna ecosystems
- HKH Region - Foothills of the Himalaya

The training datasets are hosted by SERVIR and can be downloaded from SERVIRglobal.net. The Notebooks cover the following topics with embedded exercises (and their solutions):

- • Part 1 - Getting to Know SAR Images and Forest Signatures
- • Part 2 - SAR Time Series Visualizations and Animations
- • Part 3 - Change Detection with Time Series Metrics and Log Ratio Method
- • Part 4 - SAR Time Series Change Point Detection
- • Part 5 - SAR/Optical (NDVI) Time Series Analysis
- • Part 6 - How to Make RGB Composites from Dual-Polarimetric SAR Data

Another Notebook is available that describes how to use the GDAL Virtual Raster Table (VRT) format for efficient stacking of SAR data into an analysis-ready time series data stack. All notebooks, dataset descriptions and installation instructions are also hosted on an open source GitHub repository that can be accessed from http://github.com/jkellndorfer/servir_training and http://github.com/earthbigdata/openSAR

A time series visualization QGIS plugin tool is also available on the openSAR site.

SAR Training Workshop for Forest Applications
Part 1 - Getting to Know SAR Images and Forest Signatures

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2019

This book chapter on SAR data analysis for forest applications with a focus on deforestation and forest degradation monitoring is implemented as an interactive notebook. The digital format (*a jupyter notebook*) of this chapter can readily be launched in any web browser for interactive data exploration with provided or new traning data. The notebook is comprised of text written in a combination of executable python code and markdown formatting including latex style mathematical equations. With this approach, the trainees can readily expand, change, and share the entire work with new data sets in new regions or newly available time series steps.

While we are only scratching the surface of available open source tools, the course will provide a broad overview on what modern tools can be employed for SAR focused data analysis, or remote sensing data analysis in general.

Software Installation and Data Sets

Please refer to the documents **INSTALLATION** and **DATA_HOWTO**.

The time series data sets for this training course were pre-processed with the EARTH BIG DATA *Software for Earth Big Data Processing, Prediction Modeling and Organization* (SEPPO) using cloud-based processing on Amazon Web Services. SEPPO allows for the fully automated processing of large SAR (and other remote sensing) data sets to constuct time series data effectively. The data format guide **EBD_README** explains data structures and filenaming conventions for data sets produced by EARTH BIG DATA, LLC.

Notes on Working with this Notebook

1. After launching the notebook server and opening a notebook navigate to the **Kernel** menu and choose ebd: > Kernel > Change Kernel > Python \conda env:ebd\
2. To execute code in a cell, position your blinking cursor inside a cell and either select the *Run* Button from the notebook menu bar, or use the following keystroke combination:
    - CTRL+Enter to run a cell
    - ALT+Enter to run a cell and insert a new cell below
3. To comment lines inside code cells use as first character #. You can mark several lines and use a keystroke combination to comment/uncomment the block with:
    - Windows: CTRL+/
    - MacOS: CMD+/

Importing relevant python packages

First step in the time series analysis approach after obtaining the preprocessed data stacks is the import of necessary python packages.

See the comments below as to what packages are needed and their functions. Note that all these packages should have been installed when the python anaconda environment was created.

In [1]:
```
import pandas as pd
import gdal
import numpy as np
import time,os, glob
```

In [2]:
```
%matplotlib inline
import matplotlib.pylab as plt
```

Set Project Directory and Filenames

Edit and uncomment the respective cell entries below to activate the wanted project data directory. Take a look at the EBD Data Guide: https://github.com/EarthBigData/openSAR/blob/master/documentation/EBD_DataGuide.md for an explanation of the filenaming conventions used for image and date files.

How to specify data directories:

Linux: /path/to/file

Windows: d:/path/to/file
D: is the drive letter # IMPORTANT: Always use '/' instead '\' in Windows

NOTE: Directories and filenames are specified in python as strings enclosed in single or double quotes: 'string' "string"

West Africa - Biomass Site

In [5]:
```
datadirectory='/Users/rmuench/Downloads/wa/BIOsS1'
datefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
imagefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
imagefile_cross='S32631X398020Y1315440sS1_A_vh_0001_mtfil.vrt'
```

West Africa - Niamey Deforestation Site

In [6]:
```
# datadirectory='/dev/shm/projects/c401servir/wa/cra/'
# datefile='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.dates'
# imagefile='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.vrt'
```

West Africa - Dam Site

In [7]:
```
# datadirectory='/dev/shm/projects/c401servir/wa/DAMsS1/'
# datefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.dates'
# imagefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.vrt'
```

HKH Site

In [8]:
```
# datadirectory='C:/data/hkh/time_series/S32644X696260Y3052060sS1-EBD'
# datefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.dates'
# imagefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.vrt'
# imagefile_cross='S32644X696260Y3052060sS1_D_vh_0092_mtfil.vrt'
```

Switch to the data directory

In [9]:
```
os.chdir(datadirectory)
```

In [10]:
```
os.getcwd()  # Uncomment this line to display the present working directory
```

Out[10]:
```
'/Users/rmuench/Downloads/wa/BIOsS1'
```

In [11]:
```
# glob.glob("*.vrt")   # Uncomment this line to see a List of the files
```

Acquisition Dates

Read from the dates file the dates in the time series and make a pandas date index

In [12]:
```
dates=open(datefile).readlines()
tindex=pd.DatetimeIndex(dates)
```

In [13]:
```
# From the index we make and print a lookup table for
# band numbers and dates
j=1
print('Bands and dates for',imagefile)
for i in tindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt
   1 2015-03-22    2 2015-04-03    3 2015-04-15    4 2015-05-09    5 2015-05-21
   6 2015-06-02    7 2015-06-14    8 2015-06-26    9 2015-07-08   10 2015-07-20
  11 2015-08-01   12 2015-08-13   13 2015-08-25   14 2015-09-06   15 2015-09-18
  16 2015-09-30   17 2015-10-12   18 2015-10-24   19 2015-11-17   20 2015-11-29
  21 2015-12-11   22 2015-12-23   23 2016-01-04   24 2016-01-28   25 2016-02-09
  26 2016-03-04   27 2016-03-16   28 2016-03-28   29 2016-04-09   30 2016-04-21
  31 2016-05-03   32 2016-05-15   33 2016-05-27   34 2016-06-08   35 2016-07-02
  36 2016-07-14   37 2016-07-26   38 2016-08-07   39 2016-08-19   40 2016-08-31
  41 2016-09-12   42 2016-09-24   43 2016-10-06   44 2016-10-18   45 2016-10-30
  46 2016-11-11   47 2016-11-23   48 2016-12-05   49 2016-12-17   50 2016-12-29
  51 2017-01-10   52 2017-01-22   53 2017-02-03   54 2017-02-15   55 2017-02-27
  56 2017-03-11   57 2017-03-23   58 2017-04-04   59 2017-04-16   60 2017-04-28
  61 2017-05-10   62 2017-05-22   63 2017-06-03   64 2017-06-15   65 2017-06-27
  66 2017-07-09   67 2017-07-21   68 2017-08-02   69 2017-08-14   70 2017-08-26
  71 2017-09-07   72 2017-09-19   73 2017-10-13   74 2017-10-25   75 2017-11-06
  76 2017-11-18   77 2017-11-30
```

Image data

To **open** an image file and make it readable use the gdal.Open() function. This generates an image handle that can be used for further interactions with the file:

In [14]:
```
img=gdal.Open(imagefile)
```

To explore the image, e.g. number of bands, pixels, lines you can use several functions associated with the opened image object, e.g.:

```
In [15]: print(img.RasterCount) # Number of Bands
         print(img.RasterXSize) # Number of Pixels
         print(img.RasterYSize) # Number of Lines
```

```
77
4243
3776
```

Reading data from an image band

To access any band in the image, use the img.GetRasterBand(x) function. E.g. to access the first band x=1, the last band: x=60.

```
In [16]: band=img.GetRasterBand(1)
```

Once a band is selected, several functions associated with the band are available for further processing, e.g.

- band.ReadAsArray(xoff=0,yoff=0,xsize=None,ysize=None)

So, to read the entire raster layer for the band:

```
In [17]: raster=band.ReadAsArray()
```

Subsets

Because of the potentially large data volume when dealing with time series data stacks, it may be required to read only a subset of data.

With the gdal .ReadAsArray() function, subsets can be requested with offsets and size:

img.ReadAsArray(xoff=0, yoff=0, xsize=None, ysize=None)

xoff,yoff are the offsets from the upper left corner in pixel/line coordinates.

xsize,ysize specify the size of the subset in x-direction (left to right) and y-direction (top to bottom).

E.g., to read only a **subset** of 5x5 pixels with an offset of 5 pixels and 20 lines:

```
In [18]: raster_sub=band.ReadAsArray(5,20,5,5)
```

The result is a two dimensional numpy array with the datatype the data were stored in. We can inspect these data in python by simply typing the array name on the command line:

```
In [19]: raster_sub
```

```
Out[19]: array([[4308, 4616, 4944, 4850, 4130],
                [3639, 4142, 4789, 5224, 4745],
                [3361, 3980, 4785, 5364, 4999],
                [3383, 3946, 4674, 5118, 4936],
                [3359, 3687, 4155, 4711, 5004]], dtype=uint16)
```

Displaying Bands in the Time Series of SAR Data

From the look-up table we know that bands 5 and 18 in the Niamey dataset are from late March and late October. Let's take look at these images.

HINT: Because python is an object oriented scripting language, we can often combine several steps (or function calls) into one command. See the trick below to access a raster band and read the data in one step.

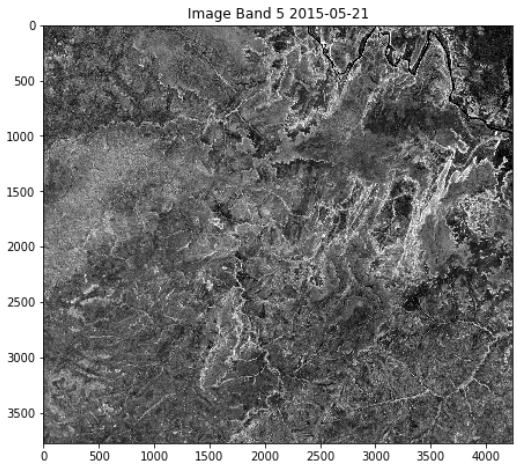
In [20]:
```
# These will select the two bands
raster_1 = img.GetRasterBand(5).ReadAsArray()
raster_2 = img.GetRasterBand(18).ReadAsArray()
```

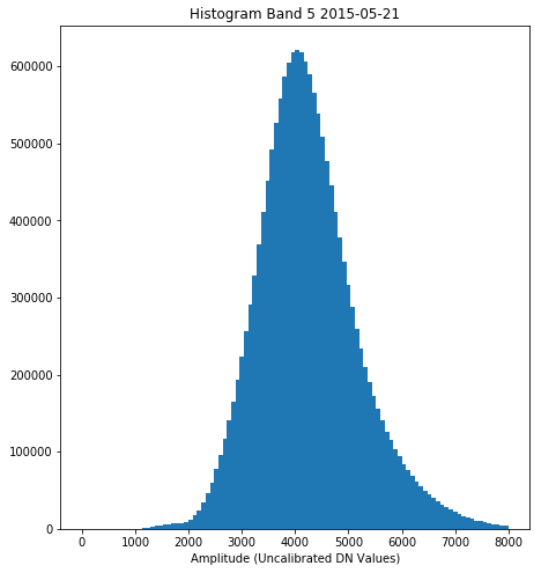
Plotting in Python to Visualize the Image Bands

Matplotlib's plotting functions allow for powerful options to display imagery. We are following some standard approaches for setting up figures.

First we are looking at a **raster band** and it's associated **histogram**.

In [21]:
```
fig = plt.figure(figsize=(16,8)) # Initialize figure with a size
ax1 = fig.add_subplot(121)  # 121 determines: 1 row, 2 plots, first plot
ax2 = fig.add_subplot(122)  # 122 determines: 1 row, 2 plots, second plot

# First plot: Image
bandnbr=5
ax1.imshow(raster_1,cmap='gray',vmin=2000,vmax=8000)
ax1.set_title('Image Band {} {}'.format(bandnbr,
                                        tindex[bandnbr-1].date()))

# Second plot: Historgram
# IMPORTANT: To get a histogram, we first need to *flatten*
# the two-dimensional image into a one-dimensional vector.
h = ax2.hist(raster_1.flatten(),bins=100,range=(0,8000))
ax2.xaxis.set_label_text('Amplitude (Uncalibrated DN Values)')
_=ax2.set_title('Histogram Band {} {}'.format(bandnbr,
                                        tindex[bandnbr-1].date()))
```





Writing a plotting function for the task

Below, the plotting commands used above are **def**ined in a function named *showImage*. Several parameters can be passed to the function, some with default values listed at the end:

- raster = a numpy two dimensional array
- tindex = a panda index array for dates
- bandnbr = the band number the corresponds to the raster
- vmin = minimim value to display
- vmax = maximum value to display

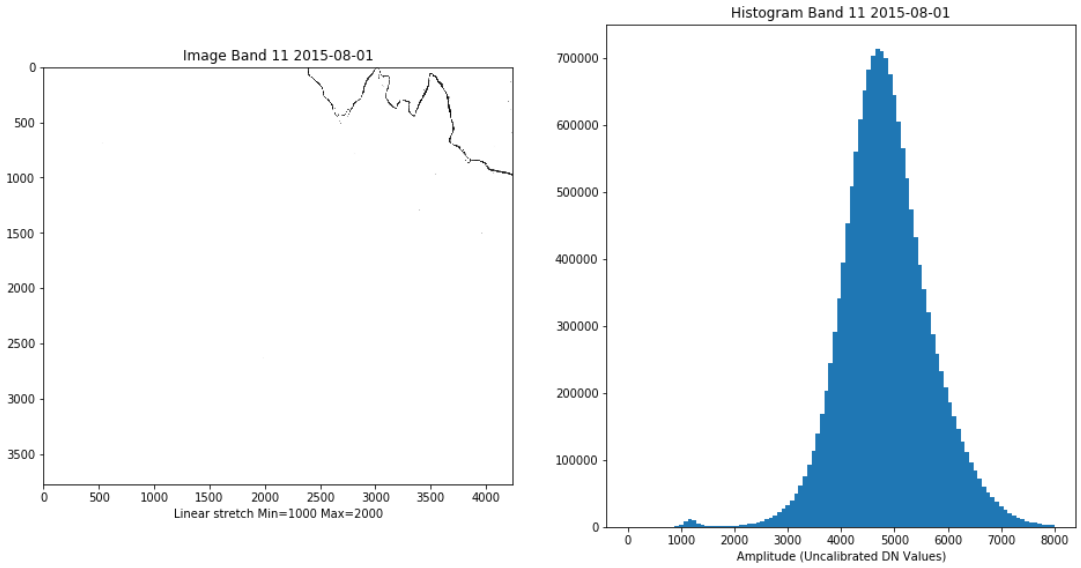
Note: By default, data will be linearly stretched between vmin and vmax.

In [22]:
```
def showImage(raster,tindex,bandnbr,vmin=None,vmax=None):
    fig = plt.figure(figsize=(16,8))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    ax1.imshow(raster,cmap='gray',vmin=vmin,vmax=vmax)
    ax1.set_title('Image Band {} {}'.format(bandnbr,
                                  tindex[bandnbr-1].date()))
    vmin=np.percentile(raster,2) if vmin==None else vmin #change vmin & vmax to
 change what values are displayed
    vmax=np.percentile(raster,98) if vmax==None else vmax
    ax1.xaxis.set_label_text(
        'Linear stretch Min={} Max={}'.format(vmin,vmax))


    h = ax2.hist(raster.flatten(),bins=100,range=(0,8000))
    ax2.xaxis.set_label_text('Amplitude (Uncalibrated DN Values)')
    ax2.set_title('Histogram Band {} {}'.format(bandnbr,
                                  tindex[bandnbr-1].date()))
```

EXERCISE 1: Read different bands and display them using the function showImage()

Use as a variable name for bands *bandnbr*. Use the already open image handle *img* to obtain the raster data from a band.

In [23]:
```
# ENTER YOUR CODE HERE
showImage(raster_2,tindex,11,1000,2000)
```



EXERCISE 2: Read two different bands and display them side by side

The output should display two bands with a heading of the band numbers. Use the concept for figures with subplots from the function showImage(). Try your code to compare images from different years and different seasons.

In [24]:
```
# ENTER YOUR CODE HERE
```

Time Series Data Stacks

Just as we can use the ReadAsArray() function on a band, we can actually use it on the entire image data stack. To read an entire stack, i.e. all bands use the function on the image data handle:

```
img.ReadAsArray()
```

CAUTION: Since this could potentially result in large memory need, it is wise to do some preliminary calcuations as to how large of a data set would be read in. For that we can do the following calculation:

$$DataVolume[GB] = \frac{N_{bands} \times N_{pixels} \times N_{lines} \times Bytes_{pixel}}{1024^3}$$

For SAR data we typically use datatypes of:

- Float 32 bit (4 bytes per pixel) for power and dB data,
- Unsigned Integer 16 bit (2 bytes per pixel) linearly scaled amplitudes, and
- Unsigned Byte (1 byte per pixel) for dB-scaled to 8 bit data

The following table gives an overview of typically used data types for SAR data analysis in python:

Data Type	Numpy Name	GDAL Name	GDAL Code	Bytes per pixel
Float 32 bit	np.float32	gdal.GDT_Float32	6	4
Unsigned Integer 16 bit	np.uint16	gdal.GDT_UInt16	2	2
Unsigned Integer 8 bit	np.uint8	gdal.GDT_Byte	1	1

Compare the result of the computation with the available RAM on the computer running the notebook.

EXERCISE 3: Compute the Data Volume of the Raster Stack

Compute the estimated data volume from the data set opened with gdal.Open() using the img object information img.RasterXSize, img.RasterYSize, img.RasterCount, img.GetRasterBand(1).DataType

In [25]:
```
# ENTER YOUR CODE HERE (if you need help see the bottom of the document)
```

Reading the SAR Time Series Subset

Let's read a image subset (offset 2000, 2000 / size 1024, 1024) of the entire time series data stack. The data are representatios of linearly scaled amplitudes scaled to unsigned 16 bit interger

We use the gdal ReadAsArray(xoff,yoff,xsize,ysize) function where:

- xoff = offset in pixels from upper left
- yoff = offset in lines from upper left
- xsize = number of pixels
- ysize = number of lines

If ReadAsArray() is called without any parameters set, the entire image data stack is read. ReadAsArray() returns a numpy array of the form:

> [bands,lines,pixels]

```
In [26]: # Alternatively you can make a subset and use
         # it in the ReadAsArray function prefixed with a star
         subset=(2000,2000,1024,1024)
         rasterDN = img.ReadAsArray(*subset)
```

The numpy .shape object tells us the dimensions of this data stack as *bands (here:time steps), lines, and pixels*:

```
In [27]: rasterDN.shape
```

```
Out[27]: (77, 1024, 1024)
```

Data conversion from linear scaled amplitudes to dB, power and amplitude data

The values of the raw image data show the linearly scaled amplitude values. These digital number (DN) values need to be converted to proper backscatter values of γ^o.

We consider conversion to dB scale (logarithmic scale) for the expression of the SAR backscatter, power, or amplitude scale.

SAR backscatter data of radiometrically terrain corrected data are often expressed as σ^o or the terrain flattened γ^o backscattering coefficients. For forest and land cover monitoring applications γ^o is the preferred metric.

Conversion from power to the logarithmic decibel (dB) scale follows:

$$\gamma^o_{dB} = 10 \times log_{10}(\gamma^o_{power})$$

As per widely used convention SAR backscatter data are often stored in 16bit unsigned integer values as linearly scaled amplitude data (referred to below as digital numbers **DN**), conversion to dB scale from the linear scaled amplitues is performed with a standard **calibration factor of -83 dB**. This is how ALOS SAR data are distributed by JAXA, how Earth Big Data LLC produces all SAR data including Sentinel-1 data, and how NISAR data will likely be scaled:

Conversion from amplitude to dB:

$$\gamma^o_{dB} = 20 * log10(DN) - 83$$

```
In [28]: rasterdB=20*np.log10(rasterDN)-83
```

Conversion from dB to power:

$$\gamma^o_{pwr} = 10^{\frac{\gamma^o_{dB}}{10}}$$

In [29]:
```
rasterPwr=np.power(10.,rasterdB/10.)
```

Conversion from power to amplitude:

$$\gamma^o_{amp} = \sqrt{\gamma^o_{pwr}}$$

In [30]:
```
rasterAmp=np.sqrt(rasterPwr)
```

Explore the image bands of the time steps

Let's explore how a band looks in the various image scales

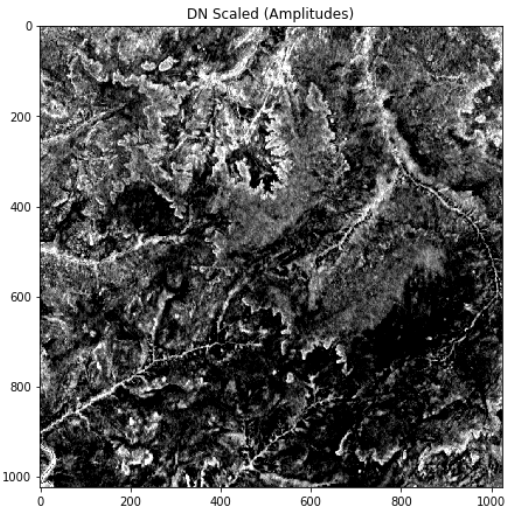
Choose the band number and find which date it is

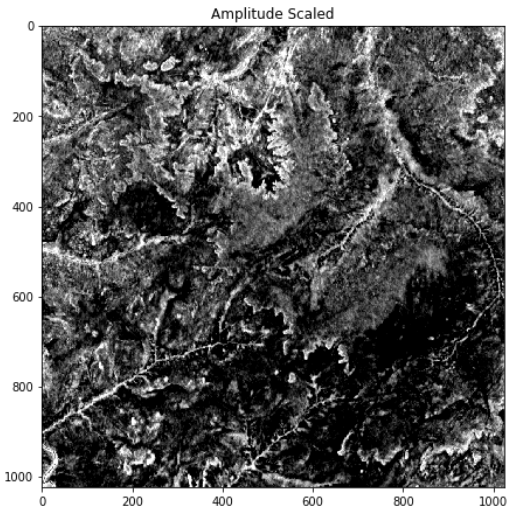
In [31]:
```
bandnbr=20
tindex[bandnbr-1]
```

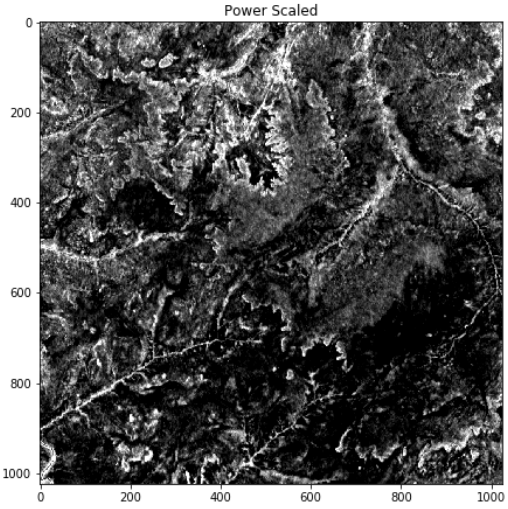
Out[31]:
```
Timestamp('2015-11-29 00:00:00')
```

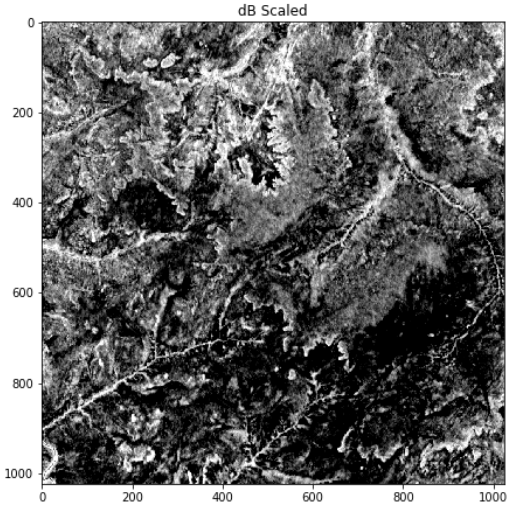
Below is the python code to create a four-part figure comparing the effect of the representation of the backscatter values in the DN, amplitude, power and dB scale.

In [32]:
```
fig=plt.figure(figsize=(16,16))

ax1=fig.add_subplot(221)
ax2=fig.add_subplot(222)
ax3=fig.add_subplot(223)
ax4=fig.add_subplot(224)

ax1.imshow(rasterDN[bandnbr],cmap='gray',
           vmin=np.percentile(rasterDN,10),
           vmax=np.percentile(rasterDN,90))
ax2.imshow(rasterAmp[bandnbr],cmap='gray',
           vmin=np.percentile(rasterAmp,10),
           vmax=np.percentile(rasterAmp,90))
ax3.imshow(rasterPwr[bandnbr],cmap='gray',
           vmin=np.percentile(rasterPwr,10),
           vmax=np.percentile(rasterPwr,90))
ax4.imshow(rasterdB[bandnbr],cmap='gray',
           vmin=np.percentile(rasterdB,10),
           vmax=np.percentile(rasterdB,90))

ax1.set_title('DN Scaled (Amplitudes)')
ax2.set_title('Amplitude Scaled')
ax3.set_title('Power Scaled')
_=ax4.set_title('dB Scaled')
```


DN Scaled (Amplitudes)
0
200
400
600
800
1000
0
200
400
600
800
1000

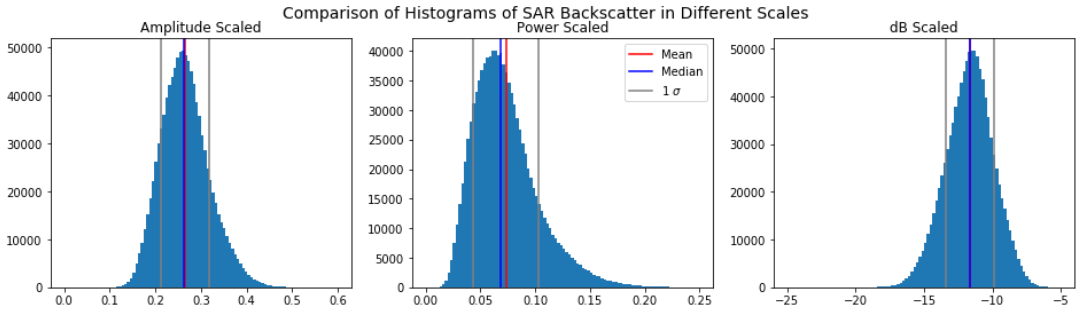

Amplitude Scaled
0
200
400
600
800
1000
0
200
400
600
800
1000


Power Scaled
0
200
400
600
800
1000
0
200
400
600
800
1000


dB Scaled
0
200
400
600
800
1000
0
200
400
600
800
1000

Comparing histograms of the amplitude, power, and dB scaled data

In [33]:
```
# Setup for three part figure
fig=plt.figure(figsize=(16,4))
fig.suptitle('Comparison of Histograms of SAR Backscatter in Different Scales',f
ontsize=14)
ax1=fig.add_subplot(131)
ax2=fig.add_subplot(132)
ax3=fig.add_subplot(133)

# Important to "flatten" the 2D raster image to produce a historgram
ax1.hist(rasterAmp[bandnbr].flatten(),bins=100,range=(0.,0.6))
ax2.hist(rasterPwr[bandnbr].flatten(),bins=100,range=(0.,0.25))
ax3.hist(rasterdB[bandnbr].flatten(),bins=100,range=(-25,-5))

# Means, medians and stddev
amp_mean=rasterAmp[bandnbr].mean()
amp_std=rasterAmp[bandnbr].std()
pwr_mean=rasterPwr[bandnbr].mean()
pwr_std=rasterPwr[bandnbr].std()
dB_mean=rasterdB[bandnbr].mean()
dB_std=rasterdB[bandnbr].std()

# Some lines for mean and median
ax1.axvline(amp_mean,color='red')
ax1.axvline(np.median(rasterAmp[bandnbr]),color='blue')
ax2.axvline(pwr_mean,color='red',label='Mean')
ax2.axvline(np.median(rasterPwr[bandnbr]),color='blue',label='Median')
ax3.axvline(dB_mean,color='red')
ax3.axvline(np.median(rasterdB[bandnbr]),color='blue')

# Lines for 1 stddev
ax1.axvline(amp_mean-amp_std,color='gray')
ax1.axvline(amp_mean+amp_std,color='gray')
ax2.axvline(pwr_mean-pwr_std,color='gray',label='1 $\sigma$')
ax2.axvline(pwr_mean+pwr_std,color='gray')
ax3.axvline(dB_mean-dB_std,color='gray')
ax3.axvline(dB_mean+dB_std,color='gray')

ax1.set_title('Amplitude Scaled')
ax2.set_title('Power Scaled')
ax3.set_title('dB Scaled')
_=ax2.legend()
```



Why is the scale important?

It is critical to use the correct scaling of SAR data for image processing operations. As we can see from the comparison of the histograms, the amplitude, power, and dB scales have different statistial distributions.

In time series analysis we often compare measurements at any given time step against the mean of the time series and compute its residuals. When we compute the mean of observations, it makes a difference whether we do that in power or dB scale. Since dB scale is a logarithmic scale, we cannot simply average data in that scale. Consider the following backscatter values and their mean:

$\gamma_1^o = -10dB$

$\gamma_2^o = -15dB$

Let's compute the mean of these values in power and dB scale and compare the result in dB scale:

In [34]:
```
g1_dB = -10
g2_dB = -15
g1_pwr = np.power(10.,-10/10.)
g2_pwr = np.power(10.,-15/10.)

mean_dB = (g1_dB+g2_dB)/2.
mean_pwr = (g1_pwr+g2_pwr)/2.
mean_pwr_inDB = 10.*np.log10(mean_pwr)

print('Mean averaging dB values         : {:.1f}'.format(mean_dB))
print('Mean averagin power values in dB : {:.1f}'.format(mean_pwr_inDB))
```

```
Mean averaging dB values         : -12.5
Mean averagin power values in dB : -11.8
```

As one can see, there is a 0.7 dB difference in the average of these two γ^o backscatter values. If we make mean estimates of backscatter values, the **correct scale** in which operations need to be performed **is the power scale.** This is critical, e.g. when speckle filters are applied, spatial operations like block averaging are performed, or time series are analyzed. Very often we implement models that relate backscatter to biophysical variables like biomass, forest height, or use thresholds to determine change. Ensure that the proper scaling is done when working with the SAR data applying these models.

Another example of the effects can be illustrated with our backscatter data from the images we extracted. Consider a 1 hectare window extracted from our data sets with an off set of 500, 500 for band 20. We compute the mean over time and space of all the pixels.

In [35]:
```
offset=500
size=5
o1=offset
o2=offset+size
```

In [36]:
```
mean_dB = rasterdB[:,o1:o2,o1:o2].mean()
mean_dB
```

Out[36]:
```
-11.302698
```

In [37]:
```
mean_pwr = rasterPwr[:,o1:o2,o1:o2].mean()
mean_pwr_in_dB = 10.* np.log10(mean_pwr)
mean_pwr_in_dB
```

Out[37]:
```
-10.75519323348999
```

As one can see, a difference of more than 0.5 dB is found simply by operating in the different scales. Hence: CAUTION!

Exploring Polarization Differences

We look at the backscatter characteristics in SAR data from like-polarized (same transmit and receive polarzation, hh or vv) and cross-polarized (vh or hv polarization). For this, we read a timestep in both polarizations, plot the histograms, and display the images in dB scale. First, we open the images, pick the bands from the same acquisition date, read the raster bands and convert them to dB scale.

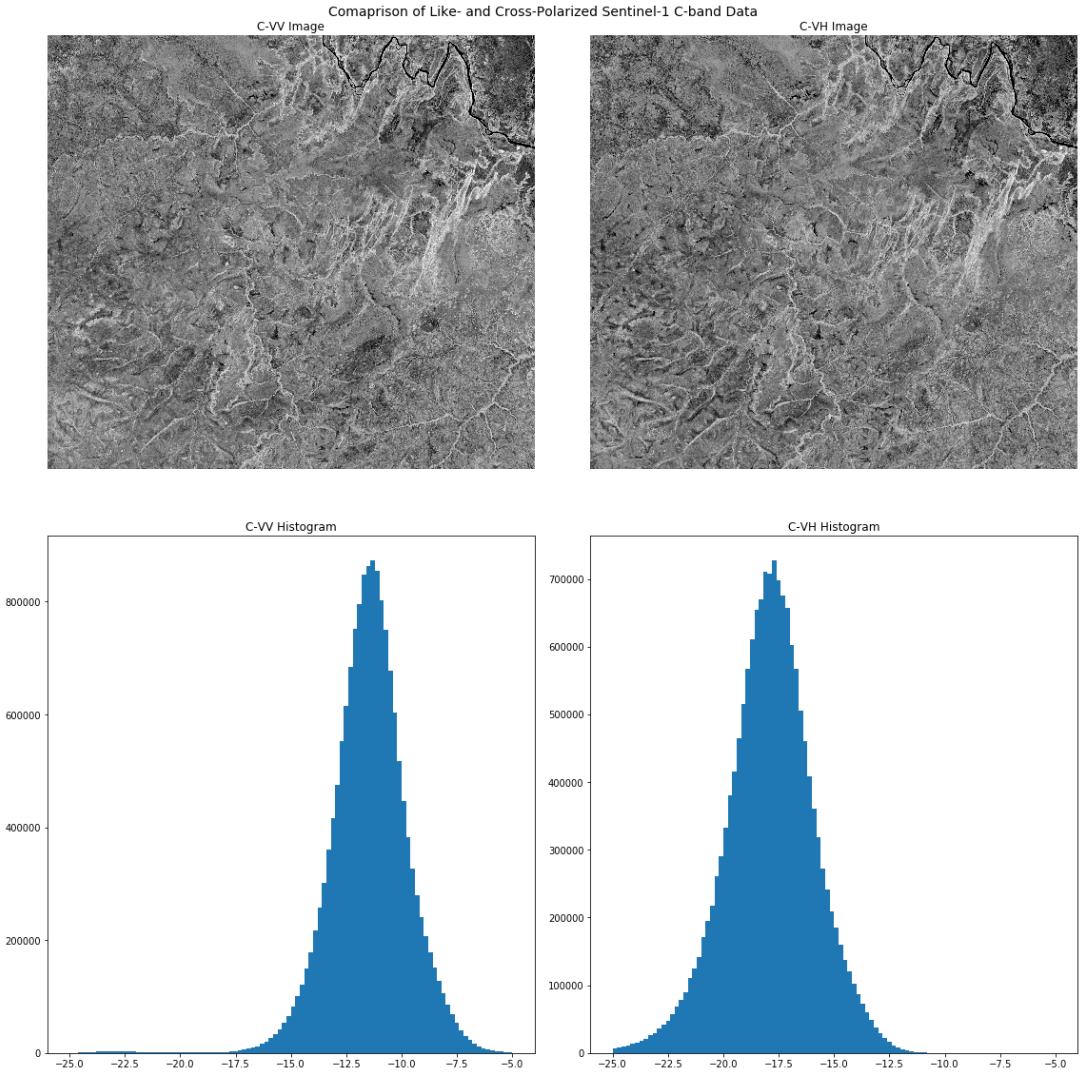
In [38]:
```
# Open the Images
img_like=gdal.Open(imagefile)
img_cross=gdal.Open(imagefile_cross)
# Pick the bands, read rasters and convert to dB
bandnbr_like=20
bandnbr_cross=20
rl=img_like.GetRasterBand(bandnbr_like).ReadAsArray()
rc=img_cross.GetRasterBand(bandnbr_cross).ReadAsArray()
rl_dB=20.*np.log10(rl)-83
rc_dB=20.*np.log10(rc)-83
```

Now, we explore the differences in the polarizations by plotting the images with their histograms. We look at the dB ranges over which the histograms spread, and can adjust the linear scaling in the image display accordingly to enhace contrast. In the case below:

- C-vv like-polarized data are mostly spread from -17.5 to -5 dB
- C-vh cross-polarized data are mostly spread from -25 to -10 dB

Thus, we note that the cross-polarized data exhibit a larger dynamic range of about 2.5 dB

In [39]:
```
fig,ax=plt.subplots(nrows=2,ncols=2,figsize=(16,16))
fig.suptitle('Comaprison of Like- and Cross-Polarized Sentinel-1 C-band Data',
             fontsize=14)
ax[0][0].set_title('C-VV Image')
ax[0][1].set_title('C-VH Image')
ax[1][0].set_title('C-VV Histogram')
ax[1][1].set_title('C-VH Histogram')
ax[0][0].axis('off')
ax[0][1].axis('off')
ax[0][0].imshow(rl_dB,vmin=-17.5,vmax=-5,cmap='gray')
ax[0][1].imshow(rc_dB,vmin=-25,vmax=-10,cmap='gray')
ax[1][0].hist(rl_dB.flatten(),range=(-25,-5),bins=100)
ax[1][1].hist(rc_dB.flatten(),range=(-25,-5),bins=100)
fig.tight_layout()  # Use the tight layout to make the figure more compact
```



EXERCISE 4: Explore different Seasons in different polarizations

Change the band numbers bandnbr_like and bandnbr_cross in the cell above to explore different bands.

EXERCISE SOLUTIONS

Solution 1

In [ ]:
```
# Pick different band numbers  for the exercise.
# Adjust scaling factors and see the effect.
bandnbr=40
raster=img.GetRasterBand(bandnbr).ReadAsArray()
showImage(raster,tindex,bandnbr,4000,8000)
```

Solution 2

In [ ]:
```
# Enter your code for the exercise here.
bandnbr1=51
raster1=img.GetRasterBand(bandnbr1).ReadAsArray()

bandnbr2=66
raster2=img.GetRasterBand(bandnbr2).ReadAsArray()

fig=plt.figure(figsize=(16,8))
ax1=fig.add_subplot(121)
ax2=fig.add_subplot(122)
ax1.imshow(raster1,vmin=2000,vmax=8000,cmap='gray')
ax2.imshow(raster2,vmin=2000,vmax=8000,cmap='gray')
ax1.set_title('Band {}   Date {}'.format(bandnbr1,tindex[bandnbr1-1].date()))
_=ax2.set_title('Band {}   Date {}'.format(bandnbr2,tindex[bandnbr2-1].date()))
```

Solution 3:

In [ ]:
```
# Get the Data type
img.GetRasterBand(1).DataType
```

In [ ]:
```
#Use the lookup table for the number of bytes per pixel for this type:
bytespp=2
size=img.RasterCount*img.RasterXSize*img.RasterYSize*bytespp/(1024*1024*1024)
print('Data Volume for {}: {:.1f} Gigabytes'.format(img.GetDescription(),size))
```

SAR TRAINING WORKSHOP: Forest Applications

PART 2 - SAR TIME SERIES VISUALIZATION AND ANIMATIONS

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2019

This section introduces more sophisticated animations for time series visualization which allow us to inspect time series in more depth. Note that html animations are not exported into the pdf file, but will display interactively.

In [1]:
```
# Turn on inline presentations
%matplotlib inline
```

In [2]:
```
# Imports
import os
import time
import gdal
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches  # Needed to draw rectangles
from matplotlib import animation, rc
from IPython.display import HTML
```

West Africa - Biomass Site

In [3]:
```
#datadirectory='/dev/shm/projects/c401servir/wa/BIOsS1/'
datadirectory='C:\\Users\\loaner.SERVIRLOAN-5057.001\\Downloads\\BIOsS1\\'
#if using a PC you may need to add two forward slashes between folders

datefile ='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
imagefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
subset=None
# subset=(2000,2000,1000,1000)
# # Browse image
# # datefile ='S32631X398020Y1315440sS1_A_vh_browse.dates'
# # imagefile='S32631X398020Y1315440sS1_A_vh_browse.tif'
# subset=None
# subset=(3700,1500,500,500)
# subset=(3000,700,500,500)
```

West Africa - Niamey Deforestation Site

In [ ]:
```
# datadirectory='/Users/rmuench/Downloads/wa/cra/'
# datefile ='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.dates'
# imagefile='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.vrt'
# subset=None
```

West Africa - Dam Site

In [ ]:
```
# datadirectory='/Users/rmuench/Downloads/wa/DAMsS1/'
# datefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.dates'
# imagefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.vrt'
# subset=None
# # subset=(2000,1500,500,500)
# # subset=(1500,500,500,500)
```

HKH Site

In [ ]:
```
# datadirectory=/Users/rmuench/Downloads/hkh/time_series/S32644X696260Y3052060sS1
-EBD'
# datefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.dates'
# imagefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.vrt'
# imagefile_cross='S32644X696260Y3052060sS1_D_vh_0092_mtfil.vrt'
```

Prepare the Animations

In [4]:
```
os.chdir(datadirectory)
```

In [5]:
```
# Get the date indices via pandas
dates=open(datefile).readlines()
tindex=pd.DatetimeIndex(dates)
```

In [6]:
```
tindex
```

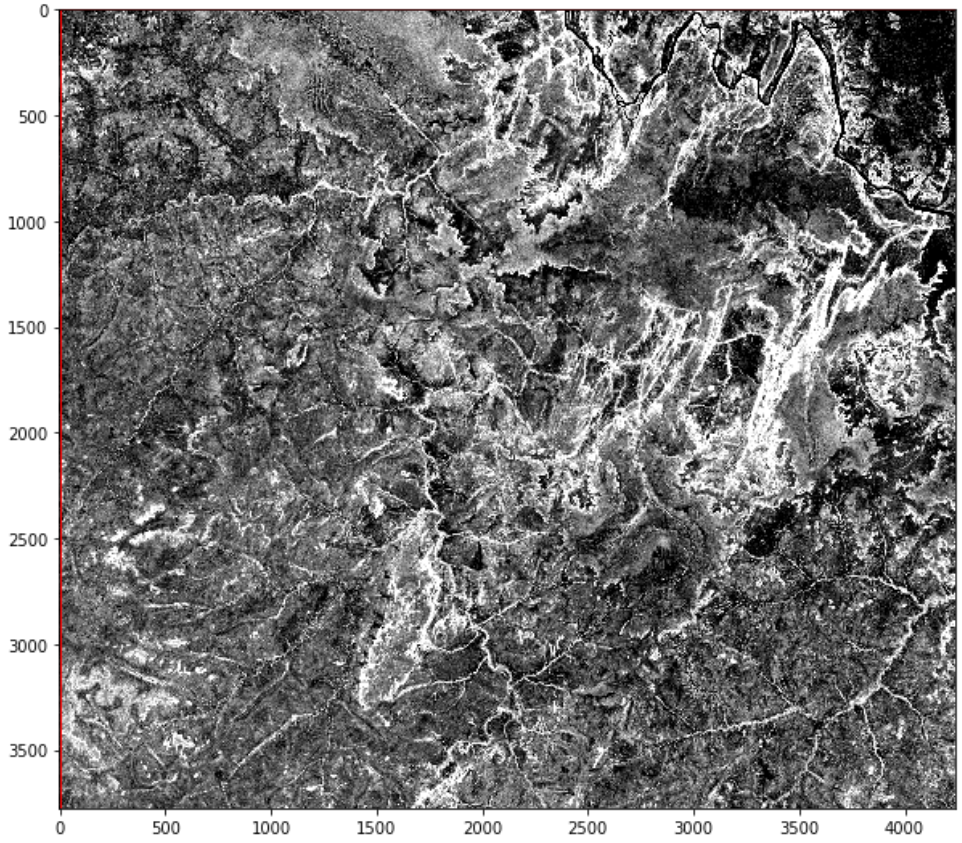
Out[6]:
```
DatetimeIndex(['2015-03-22', '2015-04-03', '2015-04-15', '2015-05-09',
               '2015-05-21', '2015-06-02', '2015-06-14', '2015-06-26',
               '2015-07-08', '2015-07-20', '2015-08-01', '2015-08-13',
               '2015-08-25', '2015-09-06', '2015-09-18', '2015-09-30',
               '2015-10-12', '2015-10-24', '2015-11-17', '2015-11-29',
               '2015-12-11', '2015-12-23', '2016-01-04', '2016-01-28',
               '2016-02-09', '2016-03-04', '2016-03-16', '2016-03-28',
               '2016-04-09', '2016-04-21', '2016-05-03', '2016-05-15',
               '2016-05-27', '2016-06-08', '2016-07-02', '2016-07-14',
               '2016-07-26', '2016-08-07', '2016-08-19', '2016-08-31',
               '2016-09-12', '2016-09-24', '2016-10-06', '2016-10-18',
               '2016-10-30', '2016-11-11', '2016-11-23', '2016-12-05',
               '2016-12-17', '2016-12-29', '2017-01-10', '2017-01-22',
               '2017-02-03', '2017-02-15', '2017-02-27', '2017-03-11',
               '2017-03-23', '2017-04-04', '2017-04-16', '2017-04-28',
               '2017-05-10', '2017-05-22', '2017-06-03', '2017-06-15',
               '2017-06-27', '2017-07-09', '2017-07-21', '2017-08-02',
               '2017-08-14', '2017-08-26', '2017-09-07', '2017-09-19',
               '2017-10-13', '2017-10-25', '2017-11-06', '2017-11-18',
               '2017-11-30'],
              dtype='datetime64[ns]', freq=None)
```

In [7]:
```
# Open the image and read the first raster band
img = gdal.Open(imagefile)
band = img.GetRasterBand(1)
# Set the subset
if subset==None:
    subset=(0,0,img.RasterXSize,img.RasterYSize)
```

In [8]:
```
subset
```

Out[8]:
```
(0, 0, 4243, 3776)
```

In [9]:
```
# Plot one band and subset outline to see which subset we are interested in
raster=band.ReadAsArray()
vmin=np.percentile(raster.flatten(),5)
vmax=np.percentile(raster.flatten(),95)
fig=plt.figure(figsize=(10,10))
ax=fig.add_subplot(111)
ax.imshow(raster,cmap='gray',vmin=vmin,vmax=vmax)
# plot the subset as rectangle
_=ax.add_patch(patches.Rectangle((subset[0],subset[1]),subset[2],subset[3],
                                 fill=False,edgecolor='red'))
```



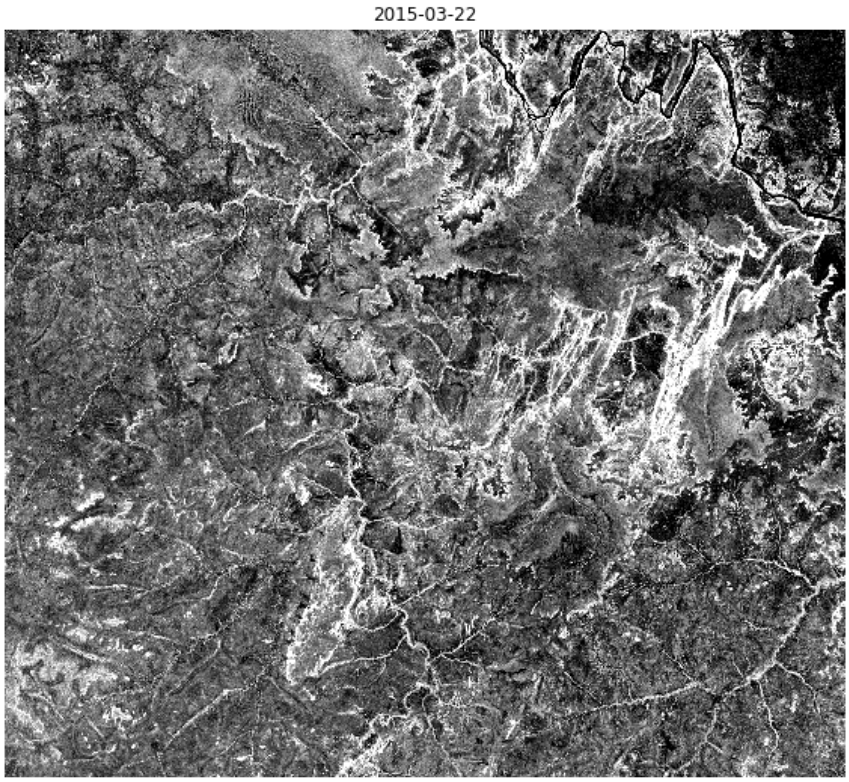
In [10]:
```
raster0 = band.ReadAsArray(*subset)
bandnbr=0 # Needed for updates
rasterstack=img.ReadAsArray(*subset)
```

In [11]:
```
%%capture
import matplotlib.pyplot as plt
import matplotlib.animation
import numpy as np

fig=plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.axis('off')
vmin=np.percentile(rasterstack.flatten(),5)
vmax=np.percentile(rasterstack.flatten(),95)

im = ax.imshow(raster0,cmap='gray',vmin=vmin,vmax=vmax)
ax.set_title("{}".format(tindex[0].date()))

def animate(i):
    ax.set_title("{}".format(tindex[i].date()))
    im.set_data(rasterstack[i])

# Interval is given in milliseconds
ani = matplotlib.animation.FuncAnimation(fig, animate,
                                         frames=rasterstack.shape[0],
                                        interval=400)
```

In [12]:
```
rc('animation',embed_limit=40971520.0)  # We need to increase the
            # limit to show the entire animation
```

In [13]:
```
from IPython.display import HTML
HTML(ani.to_jshtml())
```

Out[13]:



Plot the global means of the Time Series for the Subset

1. Conversion to power

2. Compute means

3. Convert to dB

4. Plot time series of means

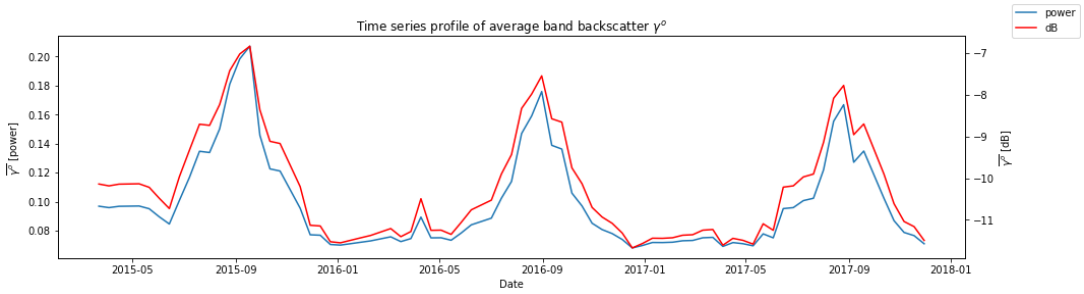
In [14]:
```
# 1. Conversion to Power
caldB=-83
calPwr = np.power(10.,caldB/10.)
rasterstack_pwr = np.power(rasterstack,2.)*calPwr
# 2. Compute Means
rs_means_pwr = np.mean(rasterstack_pwr,axis=(1,2))
# 3. Convert to dB
rs_means_dB = 10.*np.log10(rs_means_pwr)
```

In [15]:
```
rs_means_pwr.shape # Check that we got the means over time
```

Out[15]:
```
(77,)
```

In [16]:
```
# 4. Plot
fig=plt.figure(figsize=(16,4))
ax1=fig.add_subplot(111)
ax1.plot(tindex,rs_means_pwr)
ax1.set_xlabel('Date')
ax1.set_ylabel('$\overline{\gamma^o}$ [power]')


ax2=ax1.twinx()
ax2.plot(tindex,rs_means_dB,color='red')
ax2.set_ylabel('$\overline{\gamma^o}$ [dB]')
fig.legend(['power','dB'],loc=1)
plt.title('Time series profile of average band backscatter $\gamma^o$ ')
```

Out[16]:
```
Text(0.5, 1.0, 'Time series profile of average band backscatter $\\gamma^o$ ')
```



In [17]:
```
a =pd.Series(rs_means_dB,index=tindex)
```

In [18]:
```
#This will print a list of the dates and the respective dB mean values
a
```

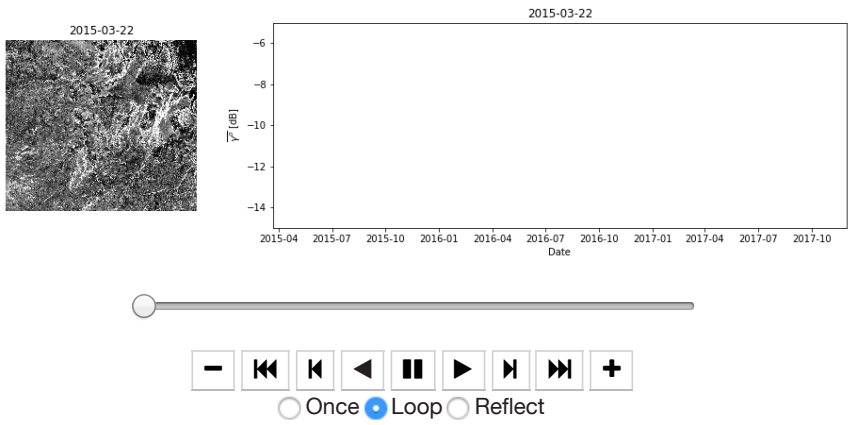
Out[18]:
```
2015-03-22   -10.139979
2015-04-03   -10.184575
2015-04-15   -10.143337
2015-05-09   -10.134364
2015-05-21   -10.218999
2015-06-02   -10.481450
2015-06-14   -10.728489
2015-06-26    -9.964857
2015-07-08    -9.330598
2015-07-20    -8.706461
2015-08-01    -8.734129
2015-08-13    -8.235253
2015-08-25    -7.423883
2015-09-06    -7.023914
2015-09-18    -6.836782
2015-09-30    -8.363434
2015-10-12    -9.116455
2015-10-24    -9.169089
2015-11-17   -10.202442
2015-11-29   -11.128629
2015-12-11   -11.145837
2015-12-23   -11.523537
2016-01-04   -11.549477
2016-01-28   -11.430446
2016-02-09   -11.372730
2016-03-04   -11.207707
2016-03-16   -11.401895
2016-03-28   -11.278982
2016-04-09   -10.491783
2016-04-21   -11.249865
                ...
2016-12-05   -11.314669
2016-12-17   -11.671808
2016-12-29   -11.566748
2017-01-10   -11.438762
2017-01-22   -11.441662
2017-02-03   -11.427748
2017-02-15   -11.367423
2017-02-27   -11.354894
2017-03-11   -11.246345
2017-03-23   -11.229724
2017-04-04   -11.605949
2017-04-16   -11.442106
2017-04-28   -11.490215
2017-05-10   -11.580404
2017-05-22   -11.092516
2017-06-03   -11.248882
2017-06-15   -10.215062
2017-06-27   -10.183667
2017-07-09    -9.969588
2017-07-21    -9.901120
2017-08-02    -9.140317
2017-08-14    -8.085749
2017-08-26    -7.777649
2017-09-07    -8.956864
2017-09-19    -8.700844
2017-10-13    -9.901974
2017-10-25   -10.611529
2017-11-06   -11.038310
2017-11-18   -11.161617
2017-11-30   -11.492415
Length: 77, dtype: float64
```

A two part figure with moving global mean backscatter of the time series in dB

In [19]:
```
%%capture
import matplotlib.pyplot as plt
import matplotlib.animation
import numpy as np

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(16,4),gridspec_kw = {'width_ratios':[1
, 3]})

vmin=np.percentile(rasterstack.flatten(),5)
vmax=np.percentile(rasterstack.flatten(),95)
im = ax1.imshow(raster0,cmap='gray',vmin=vmin,vmax=vmax)
ax1.set_title("{}".format(tindex[0].date()))
ax1.set_axis_off()

ax2.axis([tindex[0],tindex[-1],rs_means_dB.min(),rs_means_dB.max()])
ax2.set_ylabel('$\overline{\gamma^o}$ [dB]')
ax2.set_xlabel('Date')
ax2.set_ylim((-15,-5))
l, = ax2.plot([],[])


def animate(i):
    ax1.set_title("{}".format(tindex[i].date()))
    im.set_data(rasterstack[i])
    ax2.set_title("{}".format(tindex[i].date()))
    l.set_data(tindex[:(i+1)],rs_means_dB[:(i+1)])

# Interval is given in milliseconds
ani = matplotlib.animation.FuncAnimation(fig, animate,
                                         frames=rasterstack.shape[0],
                                        interval=400)
```

In [20]:
```
from IPython.display import HTML
HTML(ani.to_jshtml())
```

Out[20]:



EXERCISE

Modify the animation function to display animation of a single pixel of your choosing.

Bonus: Add a second pixel to the right hand graph.

SAR Training Workshop for Forest Applications

PART 3 - Change Detection with Time Series Metrics and Log Ratio Method

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2018

In this chapter we introduce three methods for change detection based on

- Time series metrics 95^{th} and 5^{th} percentile difference thresholding
- Time series coefficient of variation thresholding
- Log Ratio from two image pairs

Import Python modules

In [1]:
```
import os,sys,gdal
%matplotlib inline
import matplotlib.pylab as plt
import matplotlib.patches as patches  # Needed to draw rectangles
from skimage import exposure # to enhance image display
import numpy as np
import pandas as pd
```

Select the project data set and time series data

Louisiana Timber Management Site

In [ ]:
```
# SENTINEL-1 TIME SERIES STACK VV from LOUISIANA FOREST MANAGEMENT SITE
#datapath='/dev/shm/projects/c303nisar/louisiana/15SWRsS1/15SWRsS1-EBD/'
#imagefile='15SWRsS1_A_vv_0063_A_mtfil.vrt'
#datefile='15SWRsS1_A_vv_0063_A_mtfil.dates'
```

West Africa - Biomass Site

In [4]:
```
datapath='/Users/rmuench/Downloads/wa/BIOsS1'
datefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
imagefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
imagefile_cross='S32631X398020Y1315440sS1_A_vh_0001_mtfil.vrt'
```

In [5]:
```
os.chdir(datapath)
```

We are defining two helper functions for this task

- **CreateGeoTiff()** to write out images
- **timeseries_metrics()** to compute various metrics from a time series data stack

In [6]:
```
def CreateGeoTiff(Name, Array, DataType, NDV,bandnames=None,ref_image=None,
                  GeoT=None, Projection=None):
    # If it's a 2D image we fake a third dimension:
    if len(Array.shape)==2:
        Array=np.array([Array])
    if ref_image==None and (GeoT==None or Projection==None):
        raise RuntimeWarning('ref_image or settings required.')
    if bandnames != None:
        if len(bandnames) != Array.shape[0]:
            raise RuntimeError('Need {} bandnames. {} given'
                               .format(Array.shape[0],len(bandnames)))
    else:
        bandnames=['Band {}'.format(i+1) for i in range(Array.shape[0])]
    if ref_image!= None:
        refimg=gdal.Open(ref_image)
        GeoT=refimg.GetGeoTransform()
        Projection=refimg.GetProjection()
    driver= gdal.GetDriverByName('GTIFF')
    Array[np.isnan(Array)] = NDV
    DataSet = driver.Create(Name,
            Array.shape[2], Array.shape[1], Array.shape[0], DataType)
    DataSet.SetGeoTransform(GeoT)
    DataSet.SetProjection( Projection)
    for i, image in enumerate(Array, 1):
        DataSet.GetRasterBand(i).WriteArray( image )
        DataSet.GetRasterBand(i).SetNoDataValue(NDV)
        DataSet.SetDescription(bandnames[i-1])
    DataSet.FlushCache()
    return Name
```

In [7]:
```
def timeseries_metrics(raster,ndv=0):
    # Make us of numpy nan functions
    # Check if type is a float array
    if not raster.dtype.name.find('float')>-1:
        raster=raster.astype(np.float32)
    # Set ndv to nan
    if ndv != np.nan:
        raster[np.equal(raster,ndv)]=np.nan
    # Build dictionary of the metrics
    tsmetrics={}
    rperc = np.nanpercentile(raster,[5,50,95],axis=0)
    tsmetrics['mean']=np.nanmean(raster,axis=0)
    tsmetrics['max']=np.nanmax(raster,axis=0)
    tsmetrics['min']=np.nanmin(raster,axis=0)
    tsmetrics['range']=tsmetrics['max']-tsmetrics['min']
    tsmetrics['median']=rperc[1]
    tsmetrics['p5']=rperc[0]
    tsmetrics['p95']=rperc[2]
    tsmetrics['prange']=rperc[2]-rperc[0]
    tsmetrics['var']=np.nanvar(raster,axis=0)
    tsmetrics['cov']=tsmetrics['var']/tsmetrics['mean']
    return tsmetrics
```

Set the Dates

In [8]:
```
# Get the date indices via pandas
dates=open(datefile).readlines()
tindex=pd.DatetimeIndex(dates)
j=1
print('Bands and dates for',imagefile)
for i in tindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt
   1 2015-03-22    2 2015-04-03    3 2015-04-15    4 2015-05-09    5 2015-05-21
   6 2015-06-02    7 2015-06-14    8 2015-06-26    9 2015-07-08   10 2015-07-20
  11 2015-08-01   12 2015-08-13   13 2015-08-25   14 2015-09-06   15 2015-09-18
  16 2015-09-30   17 2015-10-12   18 2015-10-24   19 2015-11-17   20 2015-11-29
  21 2015-12-11   22 2015-12-23   23 2016-01-04   24 2016-01-28   25 2016-02-09
  26 2016-03-04   27 2016-03-16   28 2016-03-28   29 2016-04-09   30 2016-04-21
  31 2016-05-03   32 2016-05-15   33 2016-05-27   34 2016-06-08   35 2016-07-02
  36 2016-07-14   37 2016-07-26   38 2016-08-07   39 2016-08-19   40 2016-08-31
  41 2016-09-12   42 2016-09-24   43 2016-10-06   44 2016-10-18   45 2016-10-30
  46 2016-11-11   47 2016-11-23   48 2016-12-05   49 2016-12-17   50 2016-12-29
  51 2017-01-10   52 2017-01-22   53 2017-02-03   54 2017-02-15   55 2017-02-27
  56 2017-03-11   57 2017-03-23   58 2017-04-04   59 2017-04-16   60 2017-04-28
  61 2017-05-10   62 2017-05-22   63 2017-06-03   64 2017-06-15   65 2017-06-27
  66 2017-07-09   67 2017-07-21   68 2017-08-02   69 2017-08-14   70 2017-08-26
  71 2017-09-07   72 2017-09-19   73 2017-10-13   74 2017-10-25   75 2017-11-06
  76 2017-11-18   77 2017-11-30
```

Explore the Images

Below are a couple of plots showing the dataset

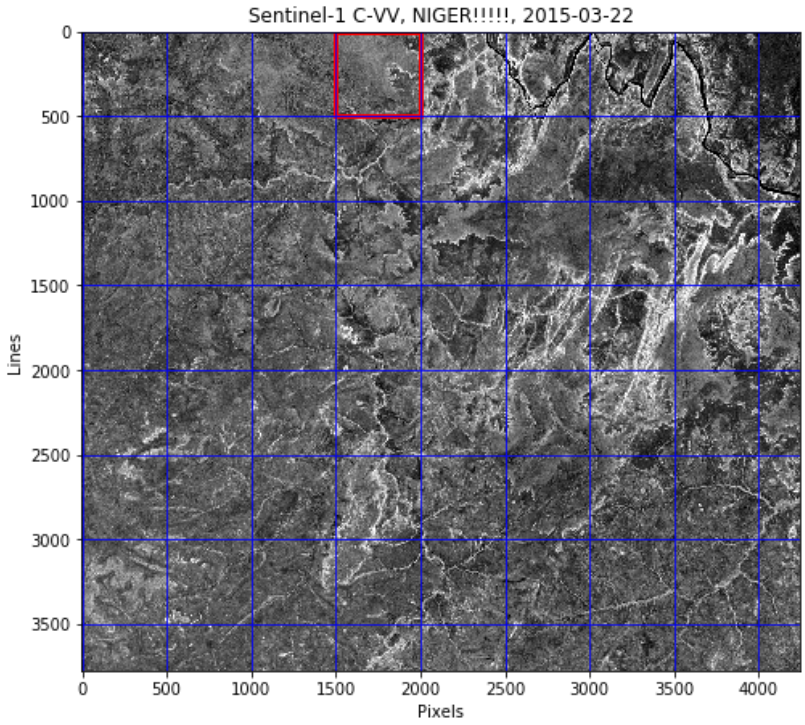
Open the image and get dimensions (bands,lines,pixels):

In [9]:
```
img=gdal.Open(imagefile)
img.RasterCount,img.RasterYSize,img.RasterXSize
```

Out[9]:
```
(77, 3776, 4243)
```

For a managable size we choose a 1000x1000 pixel subset to read the entire data stack. We also convert the amplitude data to power data right away and will perform the rest of the calculations on the power data to be mathmatically correct. NOTE: Choose a different xsize/ysize in the subset if you need to.

In [10]:
```
subset=(1500,0,500,500)   # (xoff,yoff,xsize,ysize)
bandnbr=1

rasterDN=img.GetRasterBand(bandnbr).ReadAsArray()
fig, ax = plt.subplots(figsize=(8,8))
ax.set_title('Sentinel-1 C-VV, NIGER!!!!!, {}'
             .format(tindex[bandnbr-1].date()))
ax.imshow(rasterDN,cmap='gray',vmin=2000,vmax=8000)
ax.grid(color='blue')
ax.set_xlabel('Pixels')
ax.set_ylabel('Lines')
# plot the subset as rectangle
if subset != None:
    _=ax.add_patch(patches.Rectangle((subset[0],subset[1]),
                                     subset[2],subset[3],
                                     fill=False,edgecolor='red',
                                     linewidth=3))
```



In [11]:
```
rasterDN=img.ReadAsArray(*subset)
mask=rasterDN==0
CF=np.power(10.,-8.3)
rasterPwr=np.ma.array(np.power(rasterDN,2.)*CF,mask=mask,dtype=np.float32)
# Code below is an example to generate an 8bit scaled dB image
# rasterDB=(10.*np.ma.log10(rasterPwr)+31)/0.15
# rasterDB[rasterDB<1.]=1.
# rasterDB[rasterDB>255.]=255.
# rasterDB=rasterDB.astype(np.uint8)
# rasterDB=rasterDB.filled(0)
```

We make an RGB stack to display the first, center, and last time step as a multi-temporal color composite. The np.dstack results in an array of the form [lines,pixels,bands], which is the format we need for RGB display with matplotlib's imshow() function.

Note that numpy array indexing starts with 0, so band 1 is raster[0].

In [12]:
```
rgb_bands=(1,int(img.RasterCount/2),img.RasterCount)  # first, center, last band
rgb_bands=(1,10,40)
rgb_bands=(18,45,74)
rgb_idx=np.array(rgb_bands)-1  # get array index from bands by subtracting 1
rgb=np.dstack((rasterPwr[rgb_idx[0]],rasterPwr[rgb_idx[1]],rasterPwr[rgb_idx[2
]]))
rgb_dates=(tindex[rgb_idx[0]].date(),
           tindex[rgb_idx[1]].date(),tindex[rgb_idx[2]].date())
```

We are also interested in displaying the image enhanced with histogram equalization.

We can use the function *exposure.equalize_hist()* from the skimage.exposure module

In [13]:
```
rgb_stretched=rgb.copy()
# For each band we apply the strech
for i in range(rgb_stretched.shape[2]):
    rgb_stretched[:,:,i] = exposure.\
    equalize_hist(rgb_stretched[:,:,i].data,
    mask=~np.equal(rgb_stretched[:,:,i].data,0.))
```

Now let's display the unstrechted and histogram equalized images side by side.

In [14]:
```
fig,ax = plt.subplots(1,2,figsize=(16,8))
fig.suptitle('Multi-temporal Sentinel-1 backscatter image R:{} G:{} B:{}'
             .format(rgb_dates[0],rgb_dates[1],rgb_dates[2]))
plt.axis('off')
ax[0].imshow(rgb)
ax[0].set_title('Unstreched')
ax[0].axis('off')
ax[1].imshow(rgb_stretched)
ax[1].set_title('Histogram Equalized')
_=ax[1].axis('off')
```

Computation and Visualization of the Time Series Metrics

For the entire time series, we will compute some metrics that will aid us in change detection. For each pixel in the stack we compute:

- Mean
- Median
- Maximum
- Minimum
- Range (Maximum - Minimum)
- 5th Percentile
- 95th Percentile
- PRange (95th - 5th Percentile)
- Variance
- Coefficient of Variation (Variance/Mean)

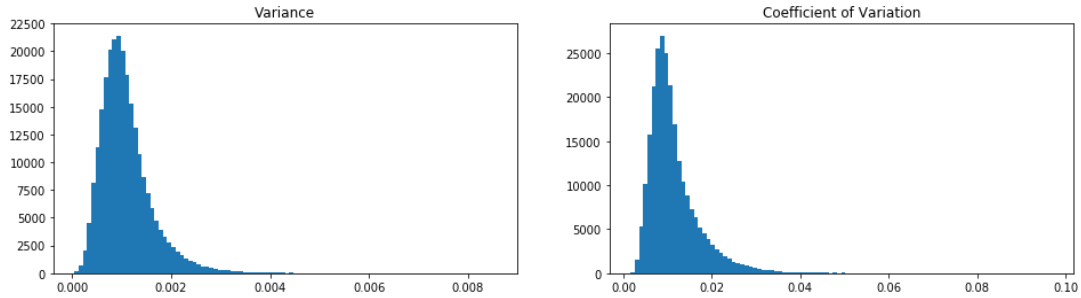
```
In [15]: metrics=timeseries_metrics(rasterPwr.filled(np.nan),ndv=np.nan)
```

```
In [16]: #Print out what the various metrics keys are
         metrics.keys()
```

```
Out[16]: dict_keys(['mean', 'max', 'min', 'range', 'median', 'p5', 'p95', 'prange', 'va
         r', 'cov'])
```

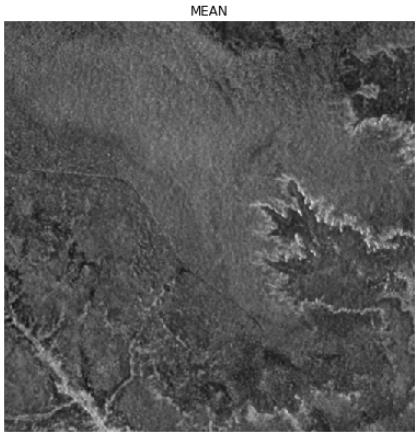
Let's look at the histograms for the time series variance and coeficient of variation to aid displaying those images:

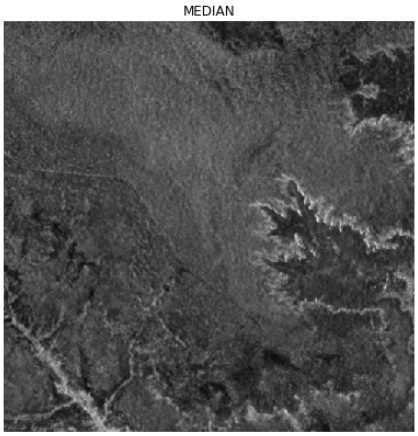
```
In [17]: fig, ax= plt.subplots(1,2,figsize=(16,4))
         ax[0].hist(metrics['var'].flatten(),bins=100)
         ax[1].hist(metrics['cov'].flatten(),bins=100)
         _=ax[0].set_title('Variance')
         _=ax[1].set_title('Coefficient of Variation')
```

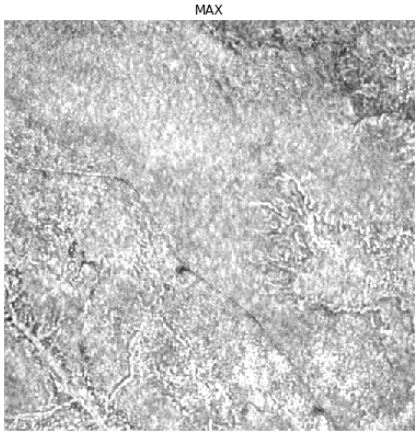


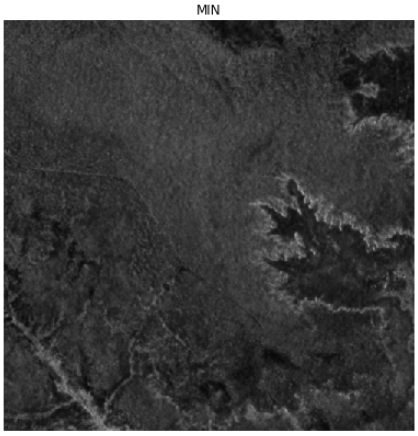
We use thresholds determined from those histograms to set the scaling in the time series visualiztion. For the backscatter metrics we choose a typical range appropriate for this ecosystem and radar sensor. A typical range is -30 dB (0.0001) to -5.2 dB (0.3).

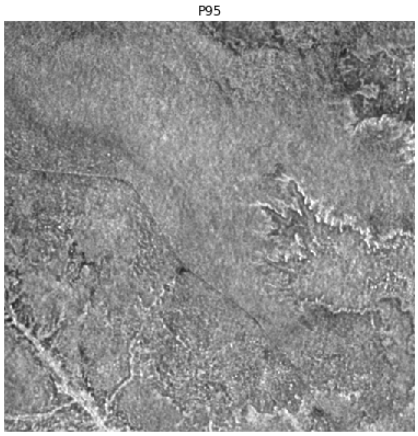
In [18]:
```
# List the metrics keys you want to plot
metric_keys=['mean', 'median', 'max', 'min',
             'p95', 'p5','range', 'prange','var','cov']
fig= plt.figure(figsize=(16,40))
idx=1
for i in metric_keys:
    ax = fig.add_subplot(5,2,idx)
    if i=='var': vmin,vmax=(0.0,0.005)
    elif i == 'cov': vmin,vmax=(0.,0.04)
    else:
        vmin,vmax=(0.0001,0.3)
    ax.imshow(metrics[i],vmin=vmin,vmax=vmax,cmap='gray')
    ax.set_title(i.upper())
    ax.axis('off')
    idx+=1
```

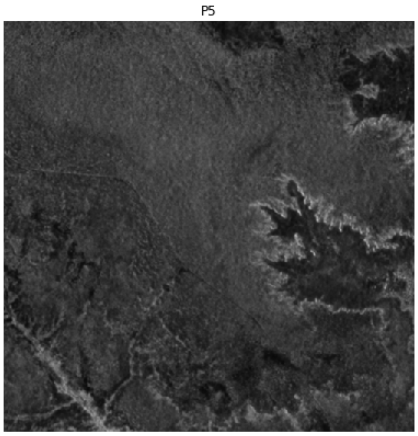

MEAN

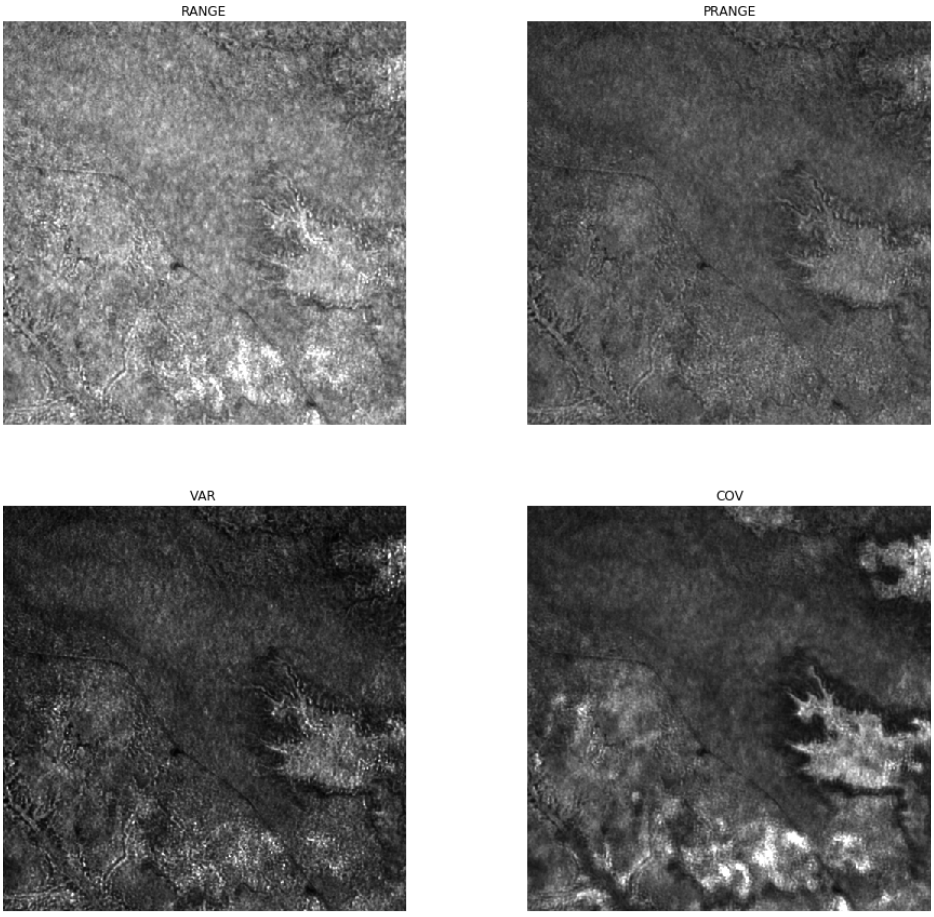

MEDIAN


MAX


MIN


P95


P5

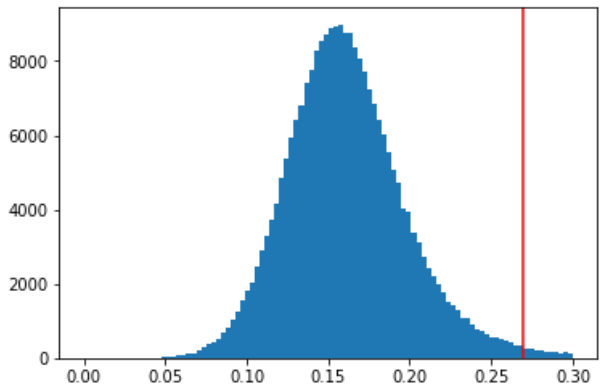


Change detection with the Percentile Difference Threshold Method

In this method we find thresholds on the 95^{th} and 5^{th} percentile difference. The advantage to look at percentiles verus maximum minus minimum is that outliers and extremas in the time series are not influencing the result.

For our example, the historgram of the 95^{th} and 5^{th} percentile difference image looks like this:

In [19]:
```
plt.hist(metrics['range'].flatten(),bins=100,range=(0,0.3))
_=plt.axvline(0.27,color='red')
```

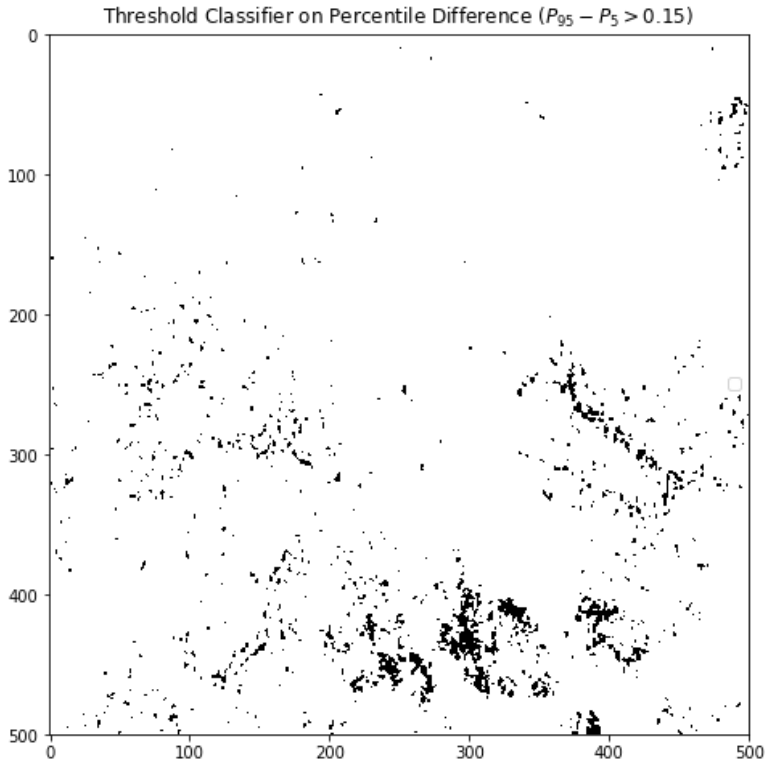


Let's visualize the change pixels (cp) where the 95th - 5th percentile difference in the time series for each pixel (x,y) exceed a threshold t

$$cp_{x,y} = P_{x,y}^{95th} - P_{x,y}^{5th} > t$$

With $t = 0.15$ the image looks like:

In [20]:
```
thres=0.25
plt.figure(figsize=(8,8))
mask=metrics['range']<thres # For display we prepare the inverse mask
maskpdiff=~mask # Store this for later output
plt.imshow(mask,cmap='gray')
plt.legend(['$p_{95} - p_5 > 0.15$'],loc='center right')
_=plt.title('Threshold Classifier on Percentile Difference ($P_{95} - P_5 > 0.15
$)')
```



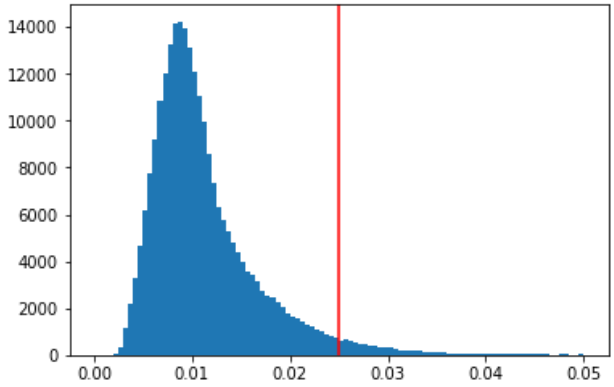
Change Detection with the Coefficient of Variation Method

We can set a threshold t for the coefficient of variation image to classify change in the time series:

$$cp_{x,y} = \frac{\sigma_{x,y}^2}{\overline{X}_{x,y}} > t$$

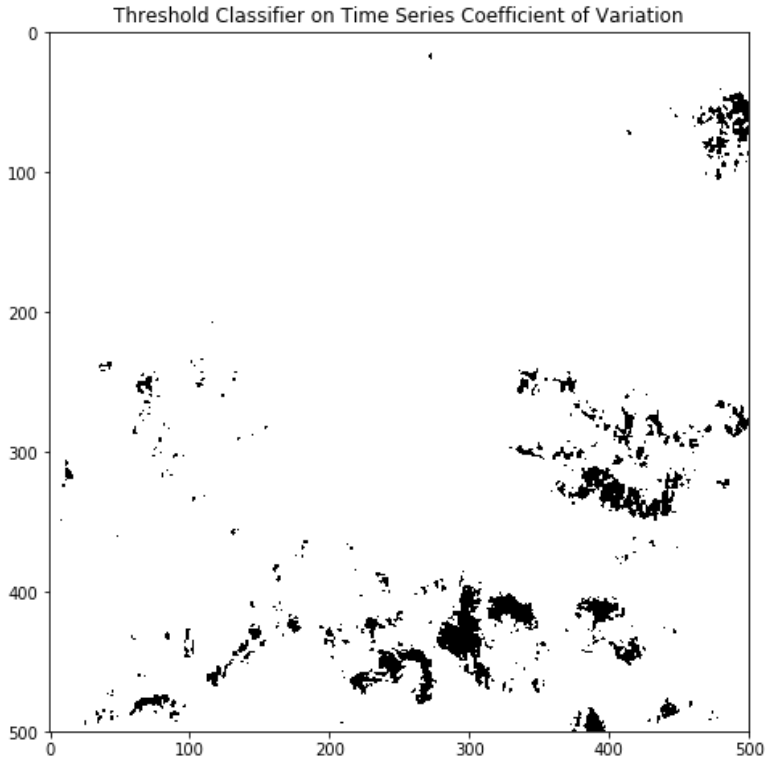
Let's look at the histogram of the coefficient of variation:

In [21]:
```
plt.hist(metrics['cov'].flatten(),bins=100,range=(0,0.05))
_=plt.axvline(0.025,color='red')
```



With a threshold t=0.01 the change pixels would look like the following image:

In [22]:
```
thres=0.025
mask=metrics['cov'] < thres
maskcv=~mask
plt.figure(figsize=(8,8))
plt.imshow(mask,cmap='gray')
_=plt.title('Threshold Classifier on Time Series Coefficient of Variation')
```



Change Detection with the Log Ratio Method

We compare two images from the same season in different years. First we look at global means of the backscatter images in the subset building a time series object of acquisition dates and global image means of backscatter.

In [23]:
```
tsmean=10*np.log10(np.nanmean(rasterPwr.filled(np.nan),axis=(1,2)))
```

We make a time series object to list the dates, mean backscatter in dB, and band index number for the rasterPwr array:

In [24]:
```
ts = pd.Series(tsmean,index=tindex)
for i in range(len(ts)):
    print(i,ts.index[i].date(),ts[i])
```

```
0 2015-03-22 -9.773781
1 2015-04-03 -9.814333
2 2015-04-15 -9.84827
3 2015-05-09 -10.075288
4 2015-05-21 -9.987606
5 2015-06-02 -9.835003
6 2015-06-14 -10.412914
7 2015-06-26 -10.64331
8 2015-07-08 -9.98234
9 2015-07-20 -9.159636
10 2015-08-01 -7.678219
11 2015-08-13 -8.60141
12 2015-08-25 -7.6070075
13 2015-09-06 -7.645421
14 2015-09-18 -6.655918
15 2015-09-30 -8.7717705
16 2015-10-12 -9.348694
17 2015-10-24 -9.547744
18 2015-11-17 -10.2138815
19 2015-11-29 -11.099142
20 2015-12-11 -11.029471
21 2015-12-23 -11.332901
22 2016-01-04 -11.346351
23 2016-01-28 -11.197915
24 2016-02-09 -11.145014
25 2016-03-04 -10.9366045
26 2016-03-16 -11.114582
27 2016-03-28 -11.000681
28 2016-04-09 -10.456753
29 2016-04-21 -11.031124
30 2016-05-03 -11.042203
31 2016-05-15 -11.248089
32 2016-05-27 -10.781347
33 2016-06-08 -10.7717905
34 2016-07-02 -10.622729
35 2016-07-14 -10.262638
36 2016-07-26 -9.969166
37 2016-08-07 -9.227007
38 2016-08-19 -8.372538
39 2016-08-31 -7.8771267
40 2016-09-12 -9.163029
41 2016-09-24 -9.04641
42 2016-10-06 -10.078144
43 2016-10-18 -10.534364
44 2016-10-30 -11.044583
45 2016-11-11 -11.120414
46 2016-11-23 -11.056729
47 2016-12-05 -11.187023
48 2016-12-17 -11.514052
49 2016-12-29 -11.376835
50 2017-01-10 -11.243304
51 2017-01-22 -11.204616
52 2017-02-03 -11.176929
53 2017-02-15 -11.093778
54 2017-02-27 -11.04459
55 2017-03-11 -10.92975
56 2017-03-23 -10.895084
57 2017-04-04 -11.270055
58 2017-04-16 -11.106432
59 2017-04-28 -11.091718
60 2017-05-10 -11.196309
61 2017-05-22 -10.516581
62 2017-06-03 -11.056223
```

```
63 2017-06-15 -10.081734
64 2017-06-27 -10.121447
65 2017-07-09 -10.04177
66 2017-07-21 -9.837778
67 2017-08-02 -9.248106
68 2017-08-14 -8.226735
69 2017-08-26 -8.096363
70 2017-09-07 -9.48612
71 2017-09-19 -9.06922
72 2017-10-13 -10.274279
73 2017-10-25 -10.869721
74 2017-11-06 -11.154692
75 2017-11-18 -11.122526
76 2017-11-30 -11.273689
```

To compare two dates for change detection with the log ratio approach we pick two dates of relative low backscatter (dry conditions) and from similar times of the year. Two such candidate dates are:

West Africa / Biomass Site example:

- 2015-11-29 -11.099142 dB (index 19)
- 2017-11-30 -11.273689 dB (index 76)

In [26]:
```
# WA biomass
Xr=rasterPwr[19]   # Reference Image
Xi=rasterPwr[76]  # New Image
```

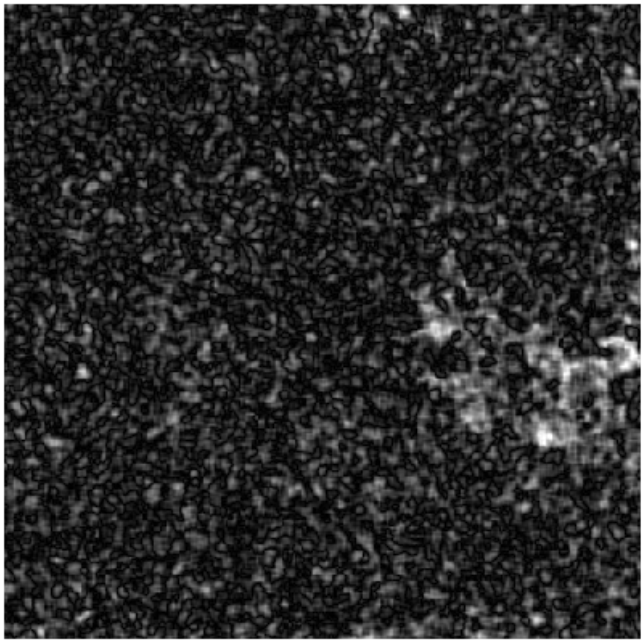
The Log ratio between the images is:

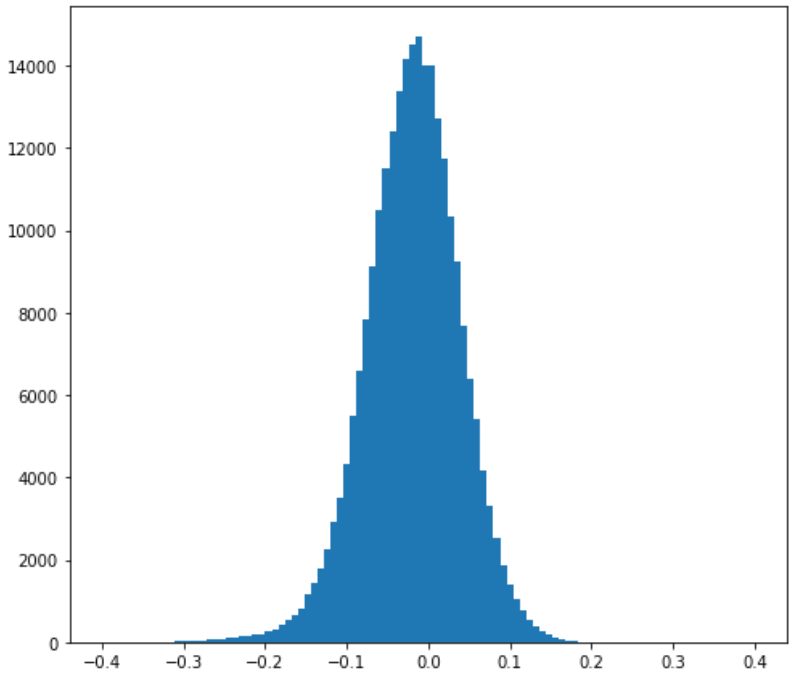
$$r = log_{10}(\frac{X_i}{X_r})$$

In [27]:
```
r = np.log10(Xi/Xr)
```

To find a threshold for change, we can display the absolute ration image $abs(r)$ and the historgram of r. We adjust the scale factors for the display to enhance visualization of change areas with largest backscatter change over the time series. Brighter values show larger change.

In [28]:
```
# Display r
fig, ax = plt.subplots(2,1,figsize=(8,16))
ax[0].axis('off')
ax[0].imshow(np.abs(r),vmin=0,vmax=0.3,cmap='gray')
_=ax[1].hist(r.flatten(),bins=100,range=(-0.4,0.4))
```





Let's define change pixels as those falling outside the range of **three** times the standard deviation of the ration image σ_r from the image mean $\overline{r}$:

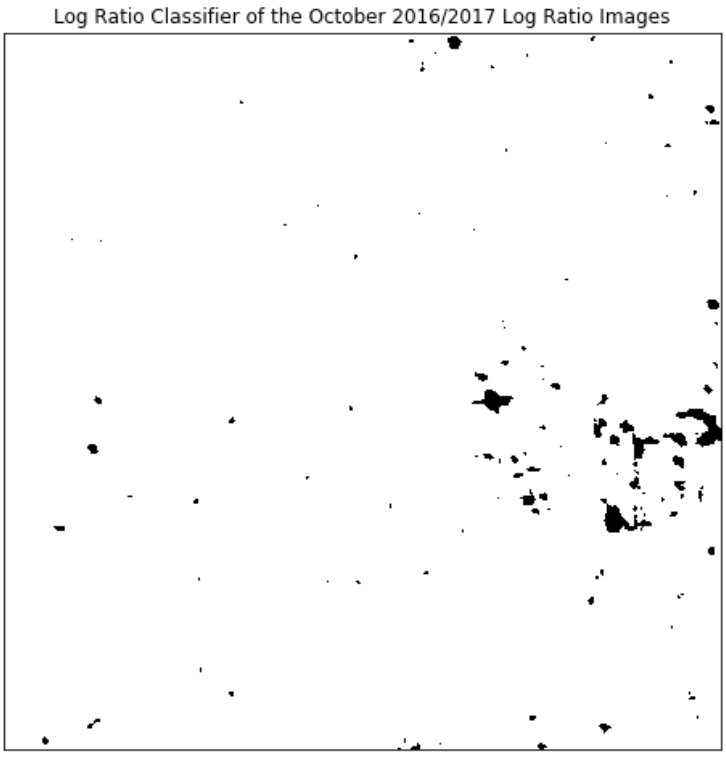
$$cp_{x,y} = (r_{x,y} < \overline{r} - 3\sigma_r) \text{ or } (r_{x,y} > \overline{r} + 3\sigma_r)$$

We are using the numpy masking to set the non-changing pixels inside the range:

In [29]:
```
stddev=np.std(r)
thres=3*stddev
mask=np.logical_and(r>-1*thres,r<thres)
masklr=~mask
```

Let's display pixels that fall outside 3 times the standard deviation

In [30]:
```
fig,ax = plt.subplots(figsize=(8,16))
ax.imshow(mask,cmap='gray')
ax.xaxis.set_ticks([])
ax.yaxis.set_ticks([])
_=ax.set_title('Log Ratio Classifier of the October 2016/2017 Log Ratio Images')
```



Write the images to an output file

Determine output geometry

First, we need to set the correct geotransformation and projection information. We retrieve the values from the input images and adjust by the subset:

In [31]:
```
proj=img.GetProjection()
geotrans=list(img.GetGeoTransform())

subset_xoff=geotrans[0]+subset[0]*geotrans[1]
subset_yoff=geotrans[3]+subset[1]*geotrans[5]
geotrans[0]=subset_xoff
geotrans[3]=subset_yoff
geotrans=tuple(geotrans)
geotrans
```

Out[31]:
```
(428020.0, 20.0, 0.0, 1390960.0, 0.0, -20.0)
```

Time series metrics images

We use the root of the time series data stack name and append a *ts*metrics_.tif ending as filenames

In [32]:
```
# Time Series Metrics as image:
# We make a new subdirectory where we will store the images
dirname=imagefile.replace('.vrt','_tsmetrics2')
os.makedirs(dirname,exist_ok=True)
print(dirname)
```

```
S32631X398020Y1315440sS1_A_vv_0001_mtfil_tsmetrics2
```

Output the individual metrics as GeoTIFF images:

In [33]:
```
Names=[] # List to keep track of all the names
for i in metrics:
    # Name, Array, DataType, NDV,bandnames=None,ref_image
    Name=os.path.join(dirname,imagefile.replace('.vrt','_'+i+'.tif'))
    CreateGeoTiff(Name,metrics[i],gdal.GDT_Float32,np.nan,[i],GeoT=geotrans,Proje
ction=proj)
    Names.append(Name)
```

Build a Virtual Raster Table on the Metrics GeoTIFF images

To tie the images in to one new raster stack of time series metrics we build a virtual raster table with all the metrics.

Trick: Use ' '.join(Names) to build one long string of names separated by a space as input to *gdalbuildvrt*

In [34]:
```
cmd='gdalbuildvrt -separate -overwrite -vrtnodata nan '+\
   dirname+'.vrt '+' '.join(Names)
# print(cmd)
os.system(cmd)
```

Out[34]:
```
0
```

In [35]:
```
os.getcwd()
```

Out[35]:
```
'/Users/rmuench/Downloads/wa/BIOsS1'
```

In [36]:
```
print('Time Series Metrics VRT File:\n',dirname+'.vrt')
```

```
Time Series Metrics VRT File:
 S32631X398020Y1315440sS1_A_vv_0001_mtfil_tsmetrics2.vrt
```

Change Images from the three methods

We are going to write one three-band GeoTIFF output file that stores the results from the three classifiers

```
In [37]: imagename=imagefile.replace('.vrt','_thresholds.tif')
         bandnames=['Percentile','COV','Log Ratio']
         Array=np.array([maskpdiff,maskcv,masklr])
         CreateGeoTiff(imagename,Array,gdal.GDT_Byte,0,bandnames,GeoT=geotrans,Projection=
         proj)
Out[37]: 'S32631X398020Y1315440sS1_A_vv_0001_mtfil_thresholds.tif'
```

This image can now be loaded into QGIS or similar programs and only the detected layers should show.

Conclusion

Thresholds for the three methods are site dependent and need to be identified with calibration data or visual post-classification interpretation, and can subsequently be adjusted to maximize classification accuracy. Also, some methods will have advantages in different scenarios.

At the Earth Big Data SEPPO Processor we actually transform many of the time series metrics data types back to lower volume storage models, e.g. 16 bit scaled amplitudes. See the EBD Data Guide below:

https://github.com/EarthBigData/openSAR/blob/master/doc/EBD_DataGuide.md

https://github.com/EarthBigData/openSAR/blob/master/doc/EBD_DataGuide.pdf

Exercises

- Change the threshold and band choices in this notebook to see the effects on detected changes.
- Load masks on the with QGIS and compare the detected areas with your time series plots and image data in QGIS.
- Look at the effect of using cross-polarized versus like-polarized polarizations

SAR Training Workshop for Forest Applications
PART 4 - SAR Time Series Change Point Detection

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2019

In this chapter we introduce the advanced concepts of change point detection in time series. One of the goals of change detection for forest applications is to identify disturbance over an observation period and the timing of events. Many tools for change point detection stem from the financial sector and are available today with different complexities. In this workbook we will analyze time series signatures from SAR with emphasis on forest time series. We will start by exploring time series at pixel levels and will work up to a change point detection scenario with image based analysis.

In [1]:
```
# Importing relevant python packages
import pandas as pd
import gdal
import numpy as np
import time,os

# For plotting
%matplotlib inline
import matplotlib.pylab as plt
import matplotlib.patches as patches

font = {'family' : 'monospace',
        'weight' : 'bold',
        'size'   : 18}
plt.rc('font',**font)
```

Set Project Directory and Filenames

West Africa - Biomass Site

In [ ]:
```
# datadirectory='c401servir/wa/BIOsS1/'
# datefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
# imagefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
```

West Africa - Niamey Deforestation Site

In [2]:
```
datadirectory='/Users/rmuench/Downloads/wa/cra/'
datefile='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.dates'
imagefile='S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.vrt'
```

West Africa - Dam Site

In [ ]:
```
# datadirectory='/dev/shm/projects/c401servir/wa/DAMsS1/'
# datefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.dates'
# imagefile='S32631X232140Y1614300sS1_A_vh_0001_A_mtfil.vrt'
```

HKH Site

In [ ]:
```
# datadirectory='hkh/time_series/S32644X696260Y3052060sS1-EBD'
# datefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.dates'
# imagefile='S32644X696260Y3052060sS1_D_vv_0092_mtfil.vrt'
# imagefile_cross='S32644X696260Y3052060sS1_D_vh_0092_mtfil.vrt'
```

In [3]:
```
# Switch to the data directory
os.chdir(datadirectory)
```

Acquisition Dates

Read from the Dates file the dates in the time series and make a pandas date index

In [4]:
```
dates=open(datefile).readlines()
tindex=pd.DatetimeIndex(dates)
j=1
print('Bands and dates for',imagefile)
for i in tindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for S32631X402380Y1491460sS1_A_vv_0001_A_mtfil.vrt
   1 2015-04-03    2 2015-11-17    3 2015-11-29    4 2015-12-11    5 2015-12-23
   6 2016-01-04    7 2016-01-28    8 2016-02-09    9 2016-03-04   10 2016-03-16
  11 2016-03-28   12 2016-04-09   13 2016-04-21   14 2016-05-03   15 2016-05-15
  16 2016-05-27   17 2016-06-08   18 2016-07-02   19 2016-07-14   20 2016-07-26
  21 2016-08-07   22 2016-08-19   23 2016-08-31   24 2016-09-12   25 2016-09-24
  26 2016-10-06   27 2016-10-18   28 2016-10-30   29 2016-11-11   30 2016-11-23
  31 2016-12-05   32 2016-12-17   33 2016-12-29   34 2017-01-10   35 2017-01-22
  36 2017-02-03   37 2017-02-15   38 2017-02-27   39 2017-03-11   40 2017-03-23
  41 2017-04-04   42 2017-04-16   43 2017-04-28   44 2017-05-10   45 2017-05-22
  46 2017-06-03   47 2017-06-15   48 2017-06-27   49 2017-07-09   50 2017-07-21
  51 2017-08-02   52 2017-08-14   53 2017-08-26   54 2017-09-07   55 2017-09-19
  56 2017-10-13   57 2017-10-25   58 2017-11-06   59 2017-11-18   60 2017-11-30
```

Image data

Get the time series raster stack from the entire training data set.

In [5]:
```
rasterstack=gdal.Open(imagefile).ReadAsArray()
```

Data Pre-Processing

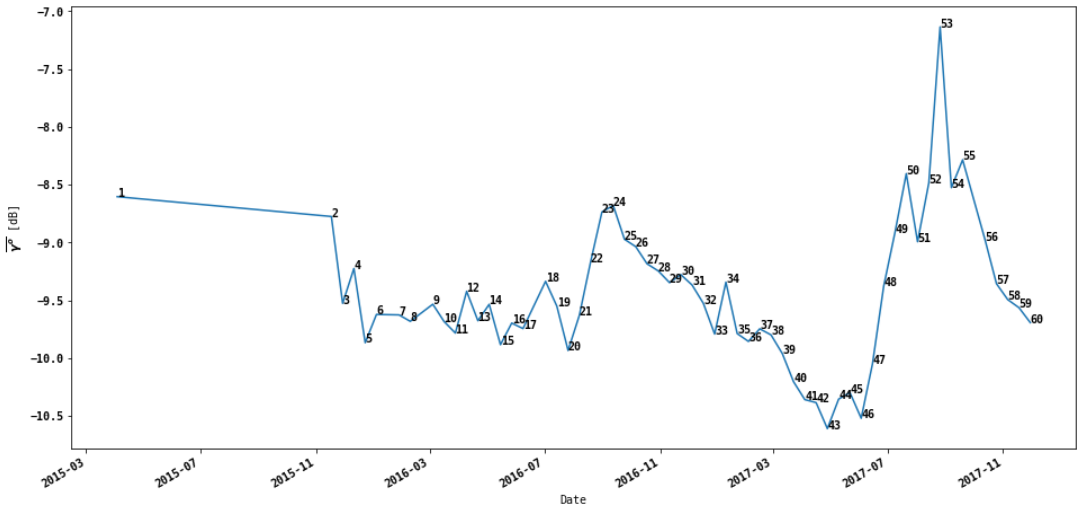
Plot the global means of the Time Series

1. Conversion to power
2. Compute means
3. Convert to dB
4. Make a pandas time series
5. Plot time series of means

In [6]:
```
# 1. Conversion to Power
caldB=-83
calPwr = np.power(10.,caldB/10.)
rasterstack_pwr = np.power(rasterstack,2.)*calPwr
# 2. Compute Means
rs_means_pwr = np.mean(rasterstack_pwr,axis=(1,2))
# 3. Convert to dB
rs_means_dB = 10.*np.log10(rs_means_pwr)
```

In [7]:
```
# 4. Make a pandas time series object
ts = pd.Series(rs_means_dB,index=tindex)
```

In [8]:
```
# 5. Use the pandas plot function of the time series object to plot
# Put band numbers as data point labels
plt.figure(figsize=(16,8))
ts.plot()
xl = plt.xlabel('Date')
yl = plt.ylabel('$\overline{\gamma^o}$ [dB]')
for xyb in zip(ts.index,rs_means_dB,range(1,len(ts)+1)):
    plt.annotate(xyb[2],xy=xyb[0:2])
```



EXERCISE

Look at the global means plot and determine from the tindex array at which dates you see maximum and minimum values. Are relative peaks associated with seasons?

Generate Time Series for Point Locations or Subsets

In python we can use the matrix slicing rules (Like Matlab) to obtain subsets of the data. For example to pick one pixel at a line/pixel location and obtain all band values, use:

> [:,line,pixel] notation.

Or, if we are interested in a subset at an offset location we can use:

> [:,yoffset:(yoffset+yrange),xoffset:(xoffset+xrange)]

In the section below we will learn how to generate time series plots for point locations (pixels) or areas (e.g. a 5x5 window region). To show individual bands, we define a showImage function which incorporates the matrix slicing from above.

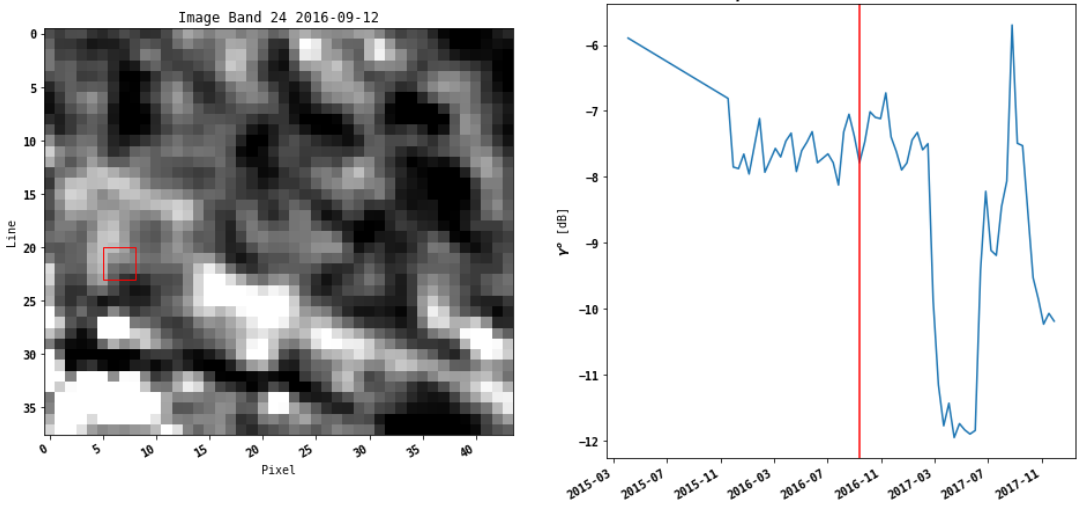
In [9]:
```
def showImage(rasterstack,tindex,bandnbr,subset=None,vmin=None,vmax=None):
    '''Input:
    rasterstack stack of images in SAR power units
    tindex time series date index
    bandnbr bandnumber of the rasterstack to dissplay'''
    fig = plt.figure(figsize=(16,8))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    # If vmin or vmax are None we use percentiles as limits:
    if vmin==None: vmin=np.percentile(rasterstack[bandnbr-1].flatten(),5)
    if vmax==None: vmax=np.percentile(rasterstack[bandnbr-1].flatten(),95)

    ax1.imshow(rasterstack[bandnbr-1],cmap='gray',vmin=vmin,vmax=vmax)
    ax1.set_title('Image Band {} {}'.format(bandnbr,tindex[bandnbr-1].date()))
    if subset== None:
        bands,ydim,xdim=rasterstack.shape
        subset=(0,0,xdim,ydim)

    ax1.add_patch(patches.Rectangle((subset[0],subset[1]),subset[2],subset[3],fi
ll=False,edgecolor='red'))
    ax1.xaxis.set_label_text('Pixel')
    ax1.yaxis.set_label_text('Line')

    ts_pwr=np.mean(rasterstack[:,subset[1]:(subset[1]+subset[3]),
                       subset[0]:(subset[0]+subset[2])],axis=(1,2))
    ts_dB=10.*np.log10(ts_pwr)
    ax2.plot(tindex,ts_dB)
    ax2.yaxis.set_label_text('$\gamma^o$ [dB]')
    ax2.set_title('$\gamma^o$ Backscatter Time Series')
    # Add a vertical line for the date where the image is displayed
    ax2.axvline(tindex[bandnbr-1],color='red')

    fig.autofmt_xdate()
```

Exercise

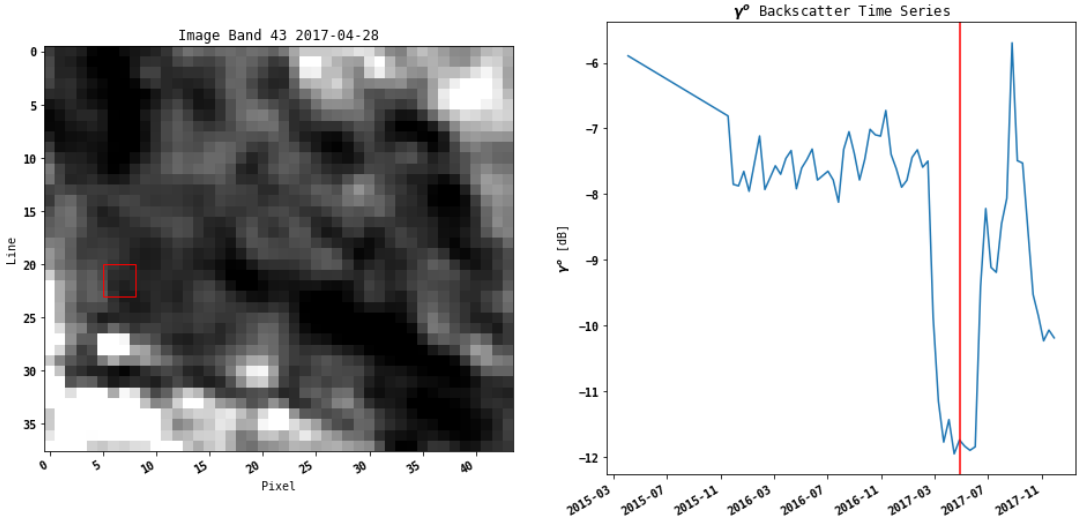
Compare band 24 and band 43 visually

In [10]:
```
bandnbr=24  #
subset=[5,20,3,3]
# subset=[30,15,3,3]
# subset=[12,10,3,3]
```

In [11]:
```
showImage(rasterstack_pwr,tindex,bandnbr,subset)
```



In [12]:
```
bandnbr=43
showImage(rasterstack_pwr,tindex,bandnbr,subset)
```



EXERCISE

For subset (5,20,3,3):

1. What are the dates where backscatter falls below - 11 dB?
2. Compute the gradients (backscatter difference between two consecutive dates.
3. What is the largest gradient of backscatter drop between two consecutive dates?
4. What are the dates associated with this gradient (before and after)?

Helper function the generate a time series object

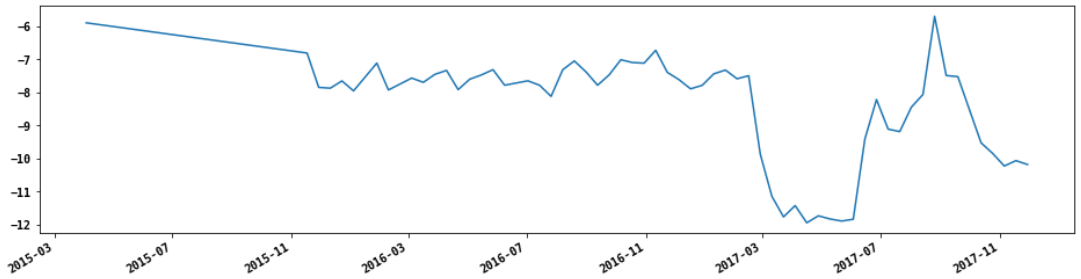
In [13]:
```
def timeSeries(rasterstack_pwr,tindex,subset,ndv=0.):
    # Extract the means along the time series axes
    # raster shape is time steps, lines, pixels.
    # With axis=1,2, we average lines and pixels for each time
    # step (axis 0)
    raster=rasterstack_pwr.copy()
    if ndv != np.nan: raster[np.equal(raster,ndv)]=np.nan
    ts_pwr=np.nanmean(raster[:,subset[1]:(subset[1]+subset[3]),
                      subset[0]:(subset[0]+subset[2])],axis=(1,2))
    # convert the means to dB
    ts_dB=10.*np.log10(ts_pwr)
    # make the pandas time series object
    ts = pd.Series(ts_dB,index=tindex)
    # return it
    return ts
```

Using the timeSeries(...) function to make a time series object for the chosen subset:

In [14]:
```
ts = timeSeries(rasterstack_pwr,tindex,subset)
```

Plot the object:

In [15]:
```
_=ts.plot(figsize=(16,4))  # _= is a trick to suppress more output.
```



ENTER YOUR CODE HERE

In [ ]:
```
# 1. What are the dates where backscatter falls below - 11 dB?
```

In [ ]:
```
# 2. Compute the gradients (backscatter difference
# between two consecutive dates.
```

In [ ]:
```
# 3. What is the largest gradient of backscatter drop
# between two consecutive dates?
```

In [ ]:
```
# What are the dates associated with this gradient
# (before and after)
```

Question: Can you field verify that change occured at this location between these two dates?

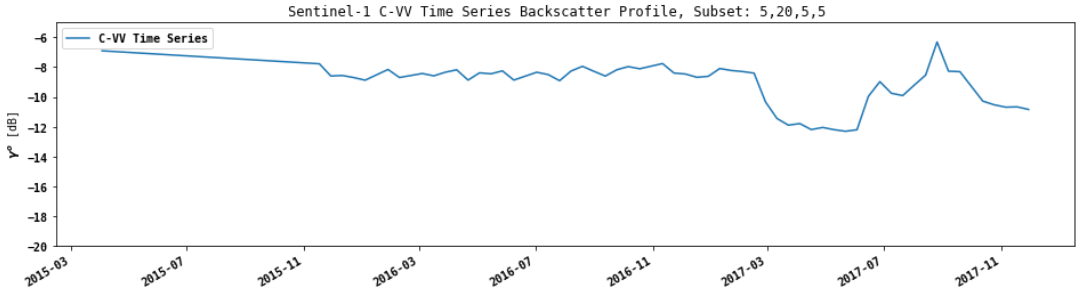
Seasonal Subsets of time series records

Let's expand upon SAR time series analysis. Often it is desirable to subset time series by season or months to compare with similar conditions of a previous year's observation. For example, in analyzing C-Band backscatter data, it might be useful to limit comparative analysis to dry season observations only as soil moisture might confuse signals during the wet seasons. In this section we will expand upon the concepts of subsetting time series along the time axis. We will make use of the pandas datatime index tools:

- Month
- Day of year

First we extract the time series again for a area at the subset location (5,20,5,5). We then convert the pandas time series to a pandas DataFrame to allow for more processing options. We also label the data value column as 'g0' for gamma0:

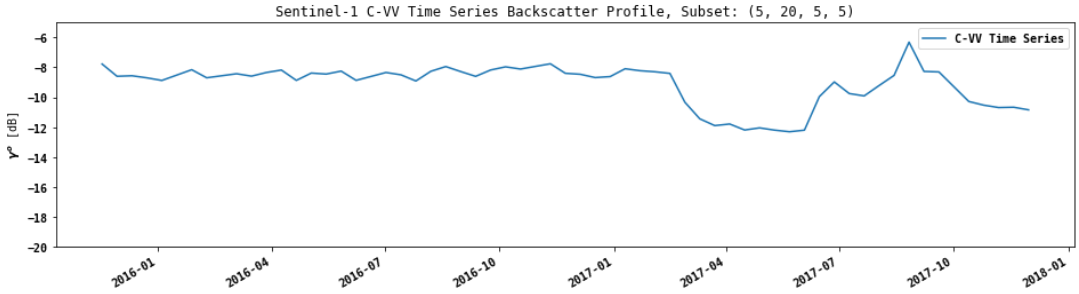
In [16]:
```
subset=(5,20,5,5)
ts = timeSeries(rasterstack_pwr,tindex,subset)
tsdf = pd.DataFrame(ts,index=ts.index,columns=['g0'])

# Plot
ylim=(-20,-5)
tsdf.plot(figsize=(16,4))
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile, Subset: 5,20,5,5  ')
plt.ylabel('$\gamma^o$ [dB]')
plt.ylim(ylim)
_=plt.legend(["C-VV Time Series"])
```



Start the time series in November 2015

We can use the pandas index parameters like month to make seasonal subsets

In [17]:
```
tsdf_sub1=tsdf[tsdf.index>'2015-11-01']

# Plot
tsdf_sub1.plot(figsize=(16,4))
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile, Subset: {}'.format(s
ubset))
plt.ylabel('$\gamma^o$ [dB]')
plt.ylim(ylim)
_=plt.legend(["C-VV Time Series"])
```

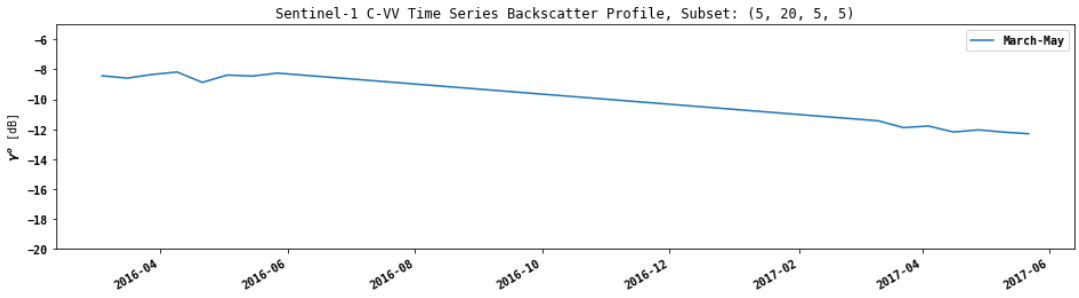


Subset by months:

We can make use of pandas DateTimeIndex object **index.month** and numpy's **logical_and** function to subset a time series easily by month.

March to May data only

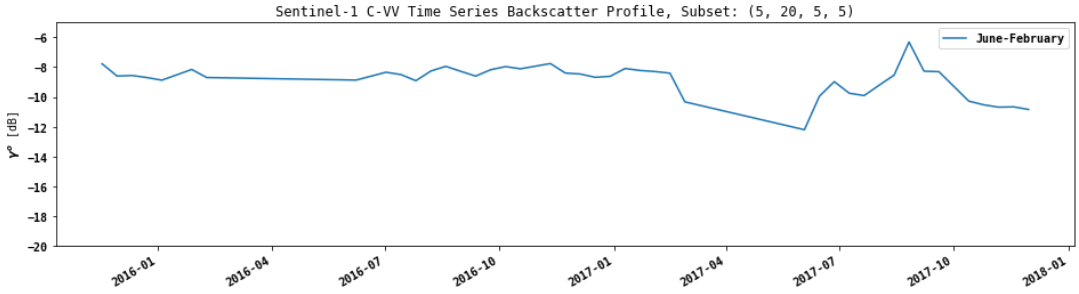
In [18]:
```
tsdf_sub2=tsdf_sub1[
    np.logical_and(tsdf_sub1.index.month>=3,tsdf_sub1.index.month<=5)]

# Plot
fig, ax = plt.subplots(figsize=(16,4))
tsdf_sub2.plot(ax=ax)
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile, Subset: {}'
          .format(subset))
plt.ylabel('$\gamma^o$ [dB]')
plt.ylim(ylim)
_=plt.legend(["March-May"])
```



All other months

Using numpy's boolean **invert** function, we can invert a selection and in this example get to all other months:

In [19]:
```
tsdf_sub3=tsdf_sub1[np.invert(
    np.logical_and(tsdf_sub1.index.month>=3,tsdf_sub1.index.month<=5))]

# Plot
fig, ax = plt.subplots(figsize=(16,4))
tsdf_sub3.plot(ax=ax)
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile, Subset: {}'
          .format(subset))
plt.ylabel('$\gamma^o$ [dB]')
plt.ylim(ylim)
_=plt.legend(["June-February"])
```

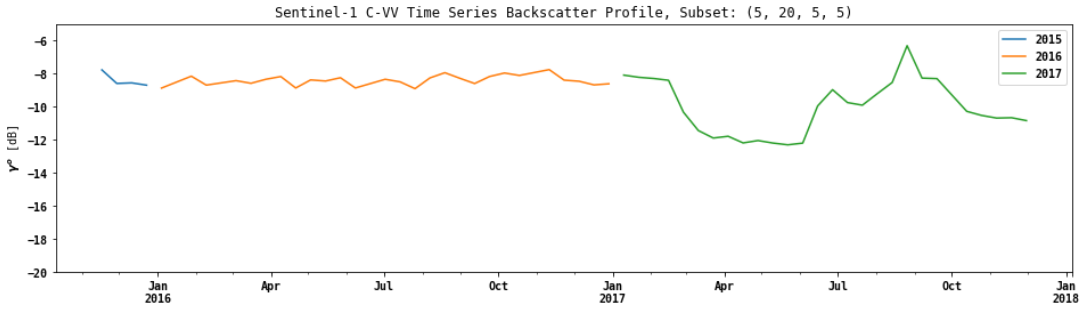


Group time series by year to compare average backscatter values

In [20]:
```
ts_sub_by_year = tsdf_sub1.groupby(pd.Grouper(freq="Y"))
```

In [21]:
```
fig, ax = plt.subplots(figsize=(16,4))
for label, df in ts_sub_by_year:
    df.g0.plot(ax=ax, label=label.year)
plt.legend()
# ts_sub_by_year.plot(ax=ax)
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile, Subset: {}'
          .format(subset))
plt.ylabel('$\gamma^o$ [dB]')
plt.ylim(ylim)
```

Out[21]: (-20, -5)



Make a pivot table to group year and sort by day of year for plotting overlapping time series

First we add two coluns to the data frame:

- Day of year (doy)
- Year

In [22]:
```
# Add doy
tsdf_sub1 = tsdf_sub1.assign(doy=tsdf_sub1.index.dayofyear)
# Add year
tsdf_sub1 = tsdf_sub1.assign(year=tsdf_sub1.index.year)
```

Then a pivot table gets created which has day of year as the index and years as columns:

In [23]:
```
piv=pd.pivot_table(tsdf_sub1,index=['doy'],columns=['year'],values=['g0'])
# Set the names for the column indices
piv.columns.set_names(['g0','Year'],inplace=True)
print(piv.head(10))
print('...\n',piv.tail(10))
```

```
g0     g0
Year 2015      2016       2017
doy
4     NaN -8.874602        NaN
10    NaN       NaN  -8.091206
22    NaN       NaN  -8.222770
28    NaN -8.155600        NaN
34    NaN       NaN  -8.294136
40    NaN -8.695752        NaN
46    NaN       NaN  -8.402759
58    NaN       NaN -10.330054
64    NaN -8.426312        NaN
70    NaN       NaN -11.441220
...
 g0          g0
Year      2015      2016       2017
doy
321  -7.774510       NaN        NaN
322        NaN       NaN -10.665520
328        NaN -8.395135        NaN
333  -8.594952       NaN        NaN
334        NaN       NaN -10.840596
340        NaN -8.461259        NaN
345  -8.560352       NaN        NaN
352        NaN -8.681982        NaN
357  -8.698992       NaN        NaN
364        NaN -8.615916        NaN
```

In [24]:
```
piv.columns.set_names(['g0','year'],inplace=True)
```

As we can see, there are NaN (Not a Number) values on the days in a year where no acquisition took place. Now we use time weighted interpolation to fill the dates for all the observations in any given year. For **time weighted interpolation** to work we need to create a dummy year as a date index, perform the interpolation, and reset the index to the day of year. This is accomplished with the following steps:

In [25]:
```
# Add fake dates for year 100 to enable time sensitive interpolation
# of missing values in the pivot table
year_doy = ['2100-{}'.format(x) for x in piv.index]
y100_doy=pd.DatetimeIndex(pd.to_datetime(year_doy,format='%Y-%j'))

# make a copy of the piv table and add two columns
piv2=piv.copy()
piv2=piv2.assign(d100=y100_doy) # add the fake year dates
piv2=piv2.assign(doy=piv2.index) # add doy as a column to replace as index later
again

# Set the index to the dummy year
piv2.set_index('d100',inplace=True,drop=True)

# PERFORM THE TIME WEIGHTED INTERPOLATION
piv2 = piv2.interpolate(method='time')  # TIME WEIGHTED INTERPOLATION!

# Set the index back to day of year.
piv2.set_index('doy',inplace=True,drop=True)
```

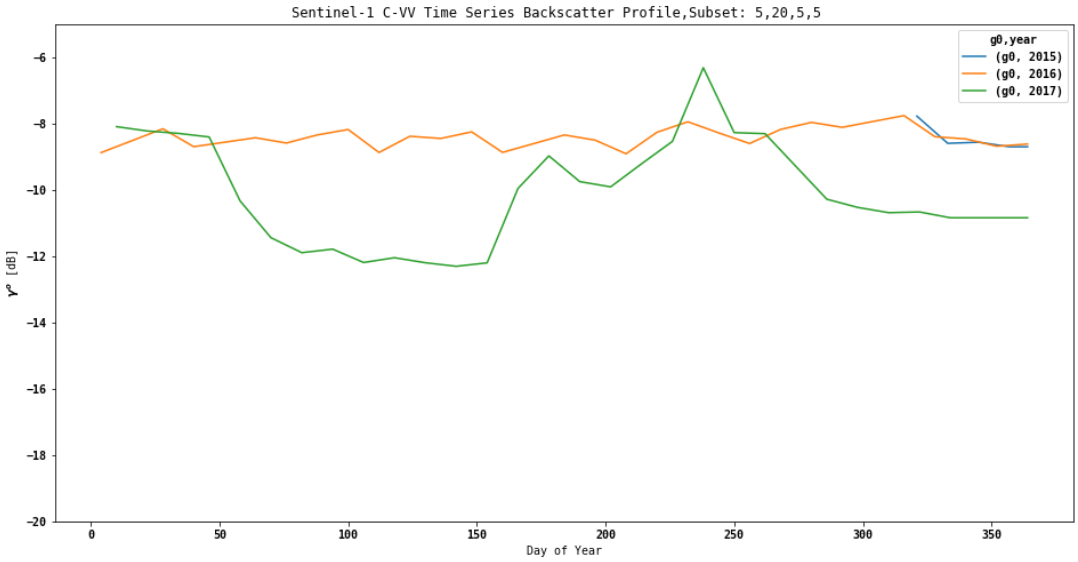
Let's inspect the new pivot table and see wheather we interpolated the NaN values where it made sense:

In [26]:
```
print(piv2.head(10))
print('...\n',piv2.tail(10))
```

```
g0     g0
year 2015      2016       2017
doy
4     NaN -8.874602        NaN
10    NaN -8.694852  -8.091206
22    NaN -8.335351  -8.222770
28    NaN -8.155600  -8.258453
34    NaN -8.425676  -8.294136
40    NaN -8.695752  -8.348448
46    NaN -8.628392  -8.402759
58    NaN -8.493672 -10.330054
64    NaN -8.426312 -10.885637
70    NaN -8.506059 -11.441220
...
 g0          g0
year      2015      2016       2017
doy
321  -7.774510 -8.023862 -10.667306
322  -7.842880 -8.076901 -10.665520
328  -8.253101 -8.395135 -10.753058
333  -8.594952 -8.422687 -10.826007
334  -8.592069 -8.428197 -10.840596
340  -8.574769 -8.461259 -10.840596
345  -8.560352 -8.553227 -10.840596
352  -8.641225 -8.681982 -10.840596
357  -8.698992 -8.654455 -10.840596
364  -8.698992 -8.615916 -10.840596
```

Now we can plot the time series data with overlapping years

In [27]:
```
piv2.plot(figsize=(16,8))
plt.title('Sentinel-1 C-VV Time Series Backscatter Profile,\
Subset: 5,20,5,5  ')
plt.ylabel('$\gamma^o$ [dB]')
plt.xlabel('Day of Year')
_=plt.ylim(ylim)
```



Change Detection on the Time Series Data

We can now analyze the time series for change. We will discuss two approaches:

1. Year-to-year differencing of the subsetted time series
2. Cumulative Sum based change detection

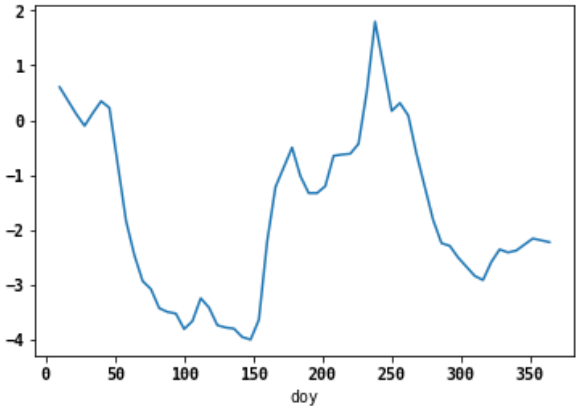
In [28]:
```
# Difference between years
# Set a dB change threshold
thres=3
```

In [29]:
```
diff1716 = (piv2.g0[2017]-piv2.g0[2016])
```

Year-to-Year Change Detection

We compute the differences between the interpolated time series and look for change with a threshold value.

In [30]:
```
_=diff1716.plot('line')
```



In [31]:
```
thres_exceeded = diff1716[abs(diff1716) > thres]
thres_exceeded
```

Out[31]:
```
doy
76    -3.081974
82    -3.430615
88    -3.499177
94    -3.527523
100   -3.810760
106   -3.665967
112   -3.248034
118   -3.420952
124   -3.741039
130   -3.782975
136   -3.803250
142   -3.956439
148   -4.003891
154   -3.642117
dtype: float64
```

From the three_exceeded dataframe we can infer the first date at which the threshold was exeeded. We would label that as a change point. As an additional criteria for labeling a change point, one can also consider the number of observations after identification of a change point where backscatter differed from the year before. If only one or two observations differed from the year before this could be considered an outlier. Addtionally, one can introduce smoothing operations with the interpolation

EXERCISE:

Work through the workbook again with selection of a different point and determine if it is a change point.

Cumulative Sums for Change Detection

Another approach to detect change in regularly acquired data is employing cumulative sums. Changes are determined against mean observations of time series. A full explanation and examples from the the financial sector can be found at http://www.variation.com/cpa/tech/changepoint.html

Time Series and Means

First let's consider a time series and it's mean observation. We look at two full years of observations from Sentinel-1 data for an area where we suspect change. In the following we consider X as a time series

$$X = (X_1, X_2, \ldots, X_n)$$

with

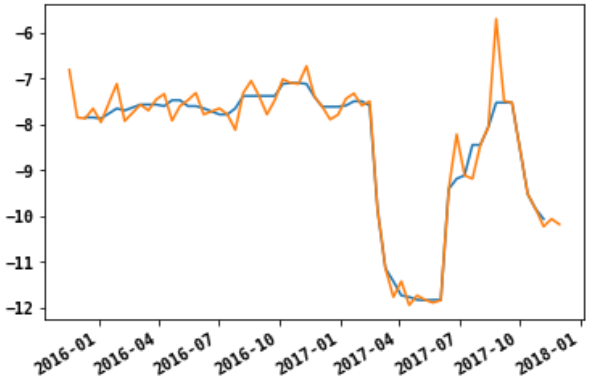
- X_i SAR backscatter at time $i = 1, \ldots, n$
- n number of observations in the time series

In [32]:
```
subset=(5,20,3,3)
#subset=(12,5,3,3)
ts1 = timeSeries(rasterstack_pwr,tindex,subset)
X = ts1[ts1.index>'2015-10-31']
```

Filtering the time series for outliers

It is advantageous in noisy SAR time series data like C-Band data to filter on the time axis. Pandas offers a "rolling" function for these purposes. With that function we can choose, for example, a median filter along the time axis. Below is an example of a median filter for an observation filters the time series when the observation before and after a time stamps are part of the filter.

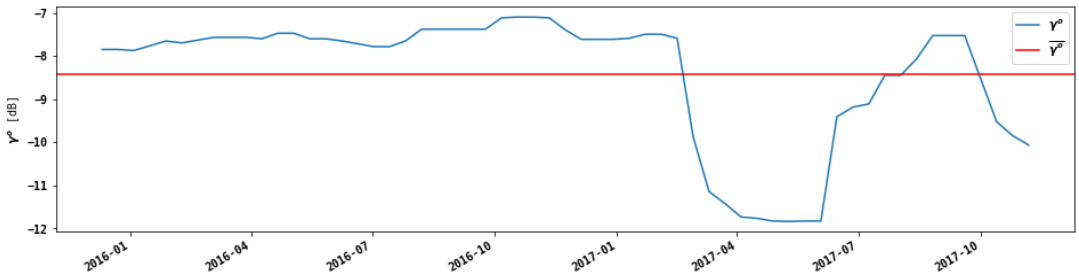
In [33]:
```
Xr=X.rolling(5,center=True).median()
Xr.plot()
_=X.plot()
```



Let's plot the time series and it's mean over the time span

In [34]:
```
X=Xr  # Uncomment if rolling mean is wanted for further computation
Xmean = X.mean()
```

In [35]:
```
fig,ax=plt.subplots(figsize=(16,4))
X.plot()
plt.ylabel('$\gamma^o$ [dB]')
ax.axhline(Xmean,color='red')
_=plt.legend(['$\gamma^o$','$\overline{\gamma^o}$'])
```



Let's determine the residuals of the time series against the mean

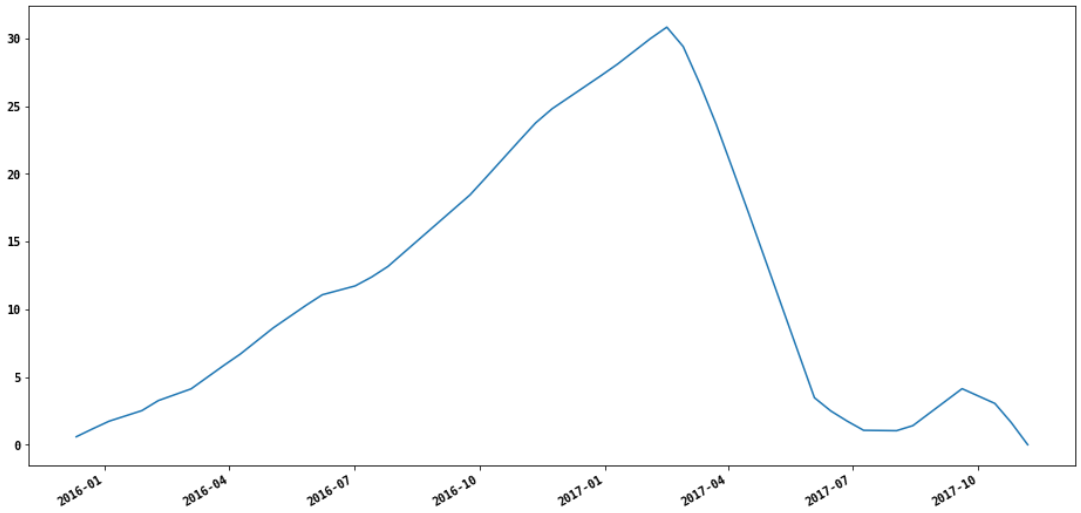
$$R = X_i - \overline{X}$$

In [36]:
```
R = X - Xmean
```

Now we compute the cumulative sum of the residuals and plot it:

$$S = \sum_{1}^{n} R_i$$

In [37]:
```
S = R.cumsum()

_=S.plot(figsize=(16,8))
```



An estimator for the magnitude of change is given as the difference between the maximum and minimum value of S

$$S_{DIFF} = S_{MAX} - S_{MIN}$$

In [38]:
```
Sdiff=S.max() - S.min()
Sdiff
```

Out[38]:
```
30.847062820803558
```

A candidate change point is identified from the S curve at the time where S_{MAX} is found:

$$T_{CP_{before}} = T(S_i = S_{MAX})$$

with

- $T_{CP_{before}}$ Timestamp of last observation before change
- S_i Cumulative Sum of R with $i = 1, \ldots n$
- n Number of observations in the time series

The first observation after change occured ($T_{CP_{after}}$) is then found as the first observation in the time series following $T_{CP_{before}}$.

For our example time series X these points are:

In [39]:
```
t_cp_before = S[S==S.max()].index[0]
print('Last date before change: {}'.format(t_cp_before.date()))
```

```
Last date before change: 2017-02-15
```

In [40]:
```
t_cp_after = S[S.index > t_cp_before].index[0]
print('First date after change: {}'.format(t_cp_after.date()))
```

```
First date after change: 2017-02-27
```

Bootstrapping the cumulative sums by randomly reordering the time series

We can determine if an identified change point is indeed a valid detection by randomly reordring the time series and comparing the various S curves. During bootstrapping we count how many times the S_{DIFF} values are greater than $S_{DIFF_{random}}$ of the identified change point. A confindence level CL is computed as:

$$CL = \frac{N_{GT}}{N_{bootstraps}}$$

with

- N_{GT} Number of times $S_{DIFF} > S_{DIFF_{random}}$
- $N_{bootstraps}$ Number of bootstraps randomizing R

Another metric for the significance of a change point is 1 minus the ratio of the mean of the $S_{DIFF_{random}}$ values and S_{DIFF}. The closer this value is approaching 1, the more significant the change point:

$$CP_{significance} = 1 - \left(\frac{\sum_{b=1}^{N_{bootstraps}} S_{DIFF_{random_i}}}{N_{bootstraps}} \Big/ S_{DIFF} \right)$$

The python code to conduct the boot strapping, including visualization of the S curves is below:

In [41]:
```
n_bootstraps=500  # bootstrap sample size
fig,ax = plt.subplots(figsize=(16,8))
S.plot(ax=ax,linewidth=3)
ax.set_ylabel('Cumulative Sums of the Residuals')
fig.legend(['S Curve for Candidate Change Point'],loc=3)
Sdiff_random_sum=0
Sdiff_random_max=0  # to keep track of the maxium Sdiff of the
                    # bootstrapped sample
n_Sdiff_gt_Sdiff_random=0  # to keep track of the maxium Sdiff of the
                    # bootstrapped sample
for i in range(n_bootstraps):
    Rrandom = R.sample(frac=1)  # Randomize the time steps of the residuals
    Srandom = Rrandom.cumsum()
    Sdiff_random=Srandom.max()-Srandom.min()
    Sdiff_random_sum += Sdiff_random
    if Sdiff_random > Sdiff_random_max:
        Sdiff_random_max = Sdiff_random
    if Sdiff > Sdiff_random:
        n_Sdiff_gt_Sdiff_random += 1
    Srandom.plot(ax=ax)
_=ax.axhline(Sdiff_random_sum/n_bootstraps)
```

In [42]:
```
CL = 1.*n_Sdiff_gt_Sdiff_random/n_bootstraps
print('Confidence Level for change point {} percent'.format(CL*100.))
```

```
Confidence Level for change point 100.0 percent
```

In [43]:
```
CP_significance = 1. - (Sdiff_random_sum/n_bootstraps)/Sdiff
print('Change point significance metric: {}'.format(CP_significance))
```

```
Change point significance metric: 0.5910152452854309
```

Another useful metric to determine strength of a change point is the normalized integral S_{ni} of the absolute values of the S curve:

$$S_{normintegral} = \frac{\int_{i=1}^{n} \frac{abs(S_i)}{\max abs(S)}}{n}$$

In [44]:
```
# NaN's to be excluded in the computation
S_ni=(S.abs()/S.abs().max()).cumsum().max()/len(S[S != np.nan])
print('Normalized Integral of cumulative sum: {}'.format(S_ni))
```

```
Normalized Integral of cumulative sum: 0.3741739515908098
```

EXERCISE

Conduct the change point analysis for different subsets in the traning data

Selection of threshold values

CL and $CP_{significance}$ can be used as threshold values for the acceptance or rejection of a candidate threshold. These values are to some degree specific to a SAR sensor and environmental conditions. E.g. L-Band SAR has a more pronounced decrease in backscatter after forest disturbance and logging, whereas C-Band can have more ambigious signals. Also moisture regime changes, e.g. with snow cover, freeze/thaw conditions or dry/wet season changes have an influence on the time series signal. For example El Nino years can suggest changes solely due to different wetting and dryup conditions pertinent to a particular year. For this reason other techniques can be added to the SAR time series ananlysis. Two techniques can readily be thought of:

- Subsetting of time series by seasons
- Detrending time series with global image means

If year-to-year comparison is the focus, the first approach likely leads to subsets that are too small for meaningful cumulative sum change point detection. The approach of interannual differencing as discussed above likely performs better.

In the following we explore the approach to detrend the data with global image means.

De-trending time series with global image means

The idea of de-trending time series with global image means should prepare time series for a somewhat more robust change point detection as global image time series anomalies stemming calibration or seasonal trends are removed prior to time series analysis. This de-trending needs to be performed with large subsets so real change is not influencing the image statistics.

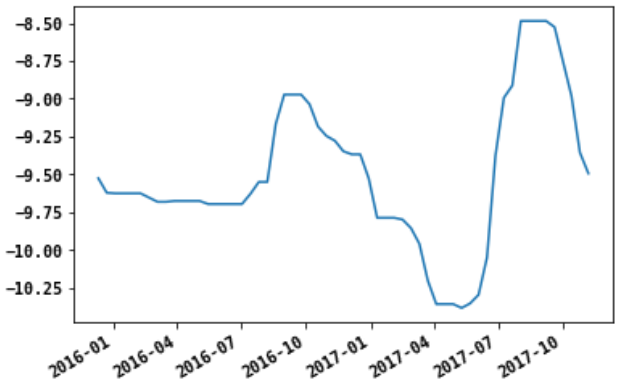
NOTE: For our small subset, we will see some of these effects.

Let's start by building a global image means time series:

In [45]:
```
means_pwr = np.mean(rasterstack_pwr,axis=(1,2))
means_dB = 10.*np.log10(means_pwr)
gm_ts = pd.Series(means_dB,index=tindex)
gm_ts=gm_ts[gm_ts.index > '2015-10-31']  # filter dates
gm_ts=gm_ts.rolling(5,center=True).median()
```

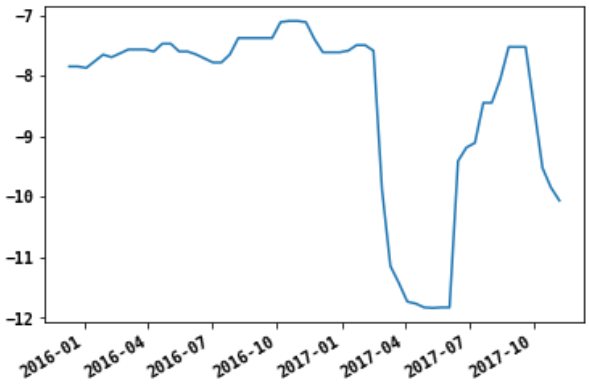
In [46]:
```
gm_ts.plot()
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x128525518>
```



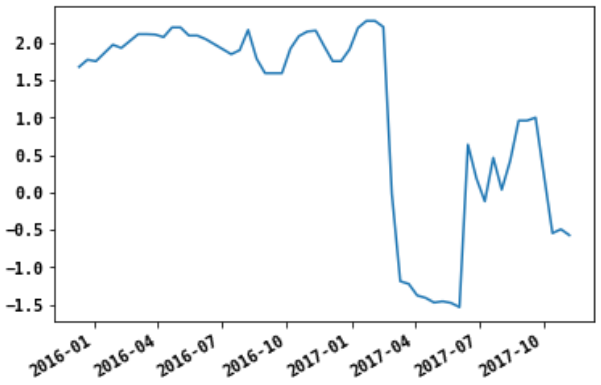
In [47]:
```
X.plot()
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12854aa90>
```



In [48]:
```
Xd=X-gm_ts
Xmean=Xd.mean()
Xd.plot()
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x128576278>
```

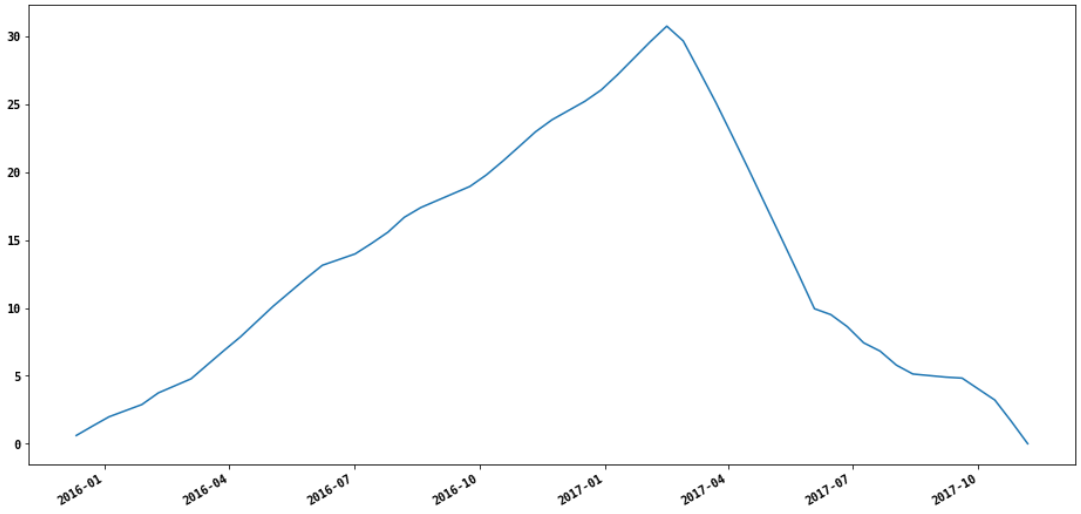


In [49]:
```
R = Xd - Xmean
```

Now we compute the cumulative sum of the residuals and plot it:

$$S = \sum_{1}^{n} R_i$$

```
In [50]: S = R.cumsum()

         _=S.plot(figsize=(16,8))
```



An estimator for the magnitude of change is given as the difference between the maximum and minimum value of S

$$S_{DIFF} = S_{MAX} - S_{MIN}$$

```
In [51]: Sdiff=S.max() - S.min()
         Sdiff
```

```
Out[51]: 30.765903052062484
```

A candidate change point is identified from the S curve at the time where S_{MAX} is found:

$$T_{CP_{before}} = T(S_i = S_{MAX})$$

with

- $T_{CP_{before}}$ Timestamp of last observation before change
- S_i Cumulative Sum of R with $i = 1, \ldots n$
- n Number of observations in the time series

The first observation after change occured ($T_{CP_{after}}$) is then found as the first observation in the time series following $T_{CP_{before}}$.

For our example time series X these points are:

In [52]:
```
t_cp_before = S[S==S.max()].index[0]
print('Last date before change: {}'.format(t_cp_before.date()))
```

```
Last date before change: 2017-02-15
```

In [53]:
```
t_cp_after = S[S.index > t_cp_before].index[0]
print('First date after change: {}'.format(t_cp_after.date()))
```

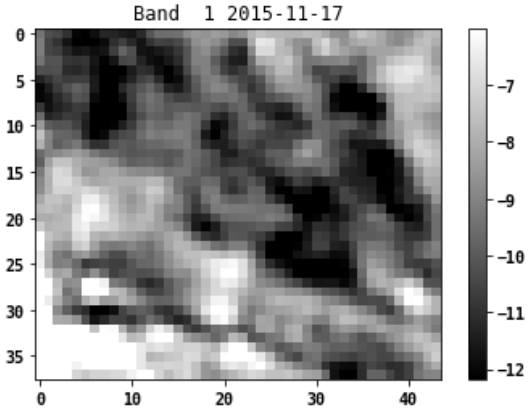
```
First date after change: 2017-02-27
```

Cumulative Sum Change Detection for the entire image

With numpy arrays we can apply the concept of cumulative sum change detection analysis effectively on the entire image stack. We take advantage of array slicing and axis-based computing in numpy. Axis 0 is the time domain in our raster stacks

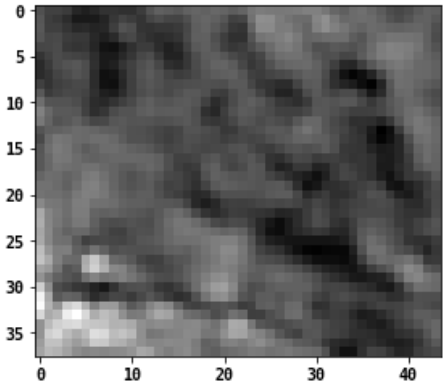
In [58]:
```
# Can do this in power or dB scale
X = rasterstack_pwr
# Filter out the first layer ( Dates >= '2015-11-1')
X_sub=X[1:,:,:]
tindex_sub=tindex[1:]
X= 10.*np.log10(X_sub)  # Uncomment to test dB scale
```

In [59]:
```
plt.figure()
#Indicate the band number
bandnbr=0
vmin=np.percentile(X[bandnbr],5)
vmax=np.percentile(X[bandnbr],95)
plt.title('Band  {} {}'.format(bandnbr+1,tindex_sub[bandnbr].date()))
plt.imshow(X[0],cmap='gray',vmin=vmin,vmax=vmax)
_=plt.colorbar()
```



In [60]:
```
Xmean=np.mean(X,axis=0)
plt.figure()
plt.imshow(Xmean,cmap='gray')
```

Out[60]:
```
<matplotlib.image.AxesImage at 0x128a80908>
```

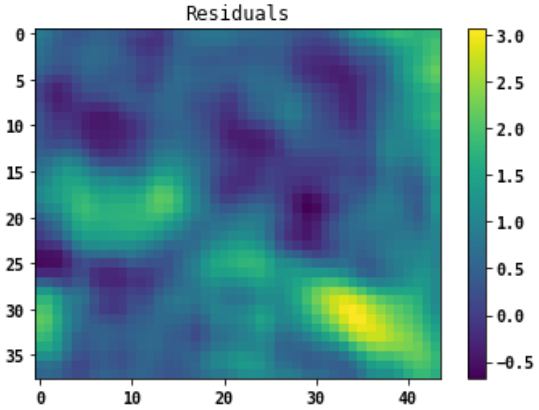


In [61]:
```
#
X.shape
```

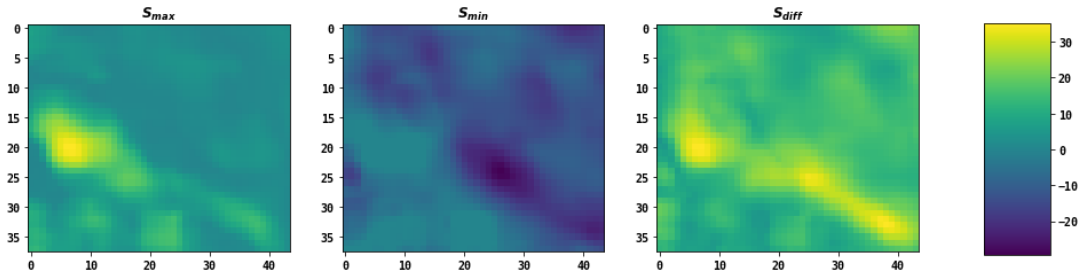
Out[61]:
```
(59, 38, 44)
```

In [62]:
```
R=X-Xmean
```

In [63]:
```
#Create an image that spatially displays the residuals (R)
plt.imshow(R[0])
plt.title('Residuals')
_=plt.colorbar()
```



In [64]:
```
S = np.cumsum(R,axis=0)
Smax= np.max(S,axis=0)
Smin= np.min(S,axis=0)
Sdiff=Smax-Smin
fig,ax=plt.subplots(1,3,figsize=(16,4))
vmin=Smin.min()
vmax=Smax.max()
p=ax[0].imshow(Smax,vmin=vmin,vmax=vmax)
ax[0].set_title('$S_{max}$')
ax[1].imshow(Smin,vmin=vmin,vmax=vmax)
ax[1].set_title('$S_{min}$')
ax[2].imshow(Sdiff,vmin=vmin,vmax=vmax)
ax[2].set_title('$S_{diff}$')
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
_=fig.colorbar(p,cax=cbar_ax)
```



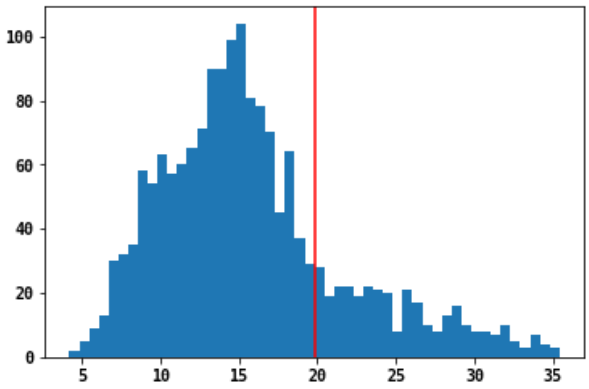
Mask Sdiff with a priori threshold for expected change

If we have an assumption as to how much actual change we expect in the image, we can threshold S_{diff} to reduce computation of the bootstrapping. For land cover change we would not expect more than 5-10% change in a landscape. So, if the test region is reasonably large, setting a threshold for expected change to 10% would be appropriate. Thus we can set a mask with the 90th percentile of the histogram of S_{diff}. In our example we'll start out with a very conservative threshold of 50%.

The histogram for S_{diff} is shown below:

In [65]:
```
#Display the Sdiff histogram
precentile=50
fig,ax=plt.subplots()
h=ax.hist(Sdiff.flatten(),bins=50)
thres=np.percentile(h[1],50)
print('At the {}% percentile, the threshold value is {:2.2f}'.format(precentile,
thres))
_=ax.axvline(thres,color='red')
```

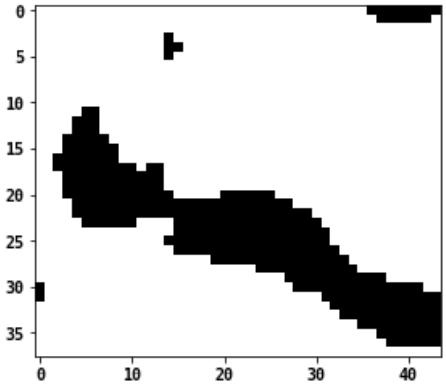
```
At the 50% percentile, the threshold value is 19.82
```



At the 50% percentile, the threshold value is ____ (printed above the histogram)

Using this threshold, we can visualize the candidate changepoints:

In [66]:
```
Sdiffmask=Sdiff<thres
_=plt.imshow(Sdiffmask,cmap='gray')
```



Now we can filter our Residuals and perform bootstrapping analysis on these data. We make use of numpy masked arrays for this purpose.

In [67]:
```
Rmask = np.broadcast_to(Sdiffmask,R.shape)
```

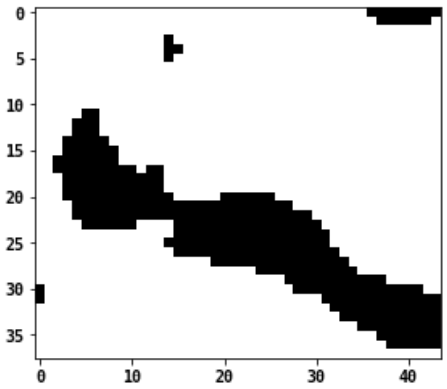
In [68]:
```
Rmasked = np.ma.array(R,mask=Rmask)
```

On the masked time series stack of residuals we can compute the cumulative sums:

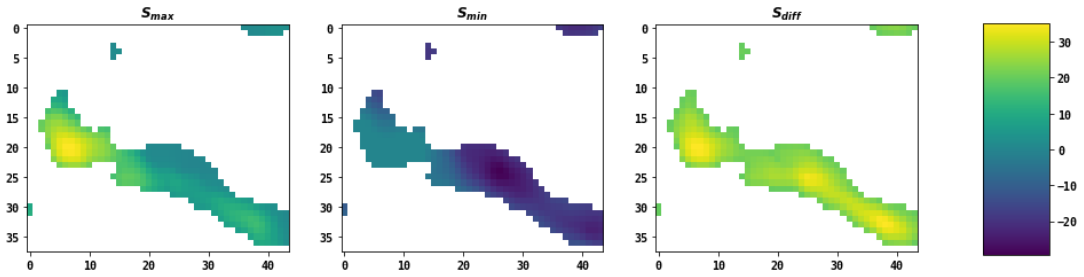
```
In [69]: Smasked = np.ma.cumsum(Rmasked,axis=0)
```

$S_{MAX}, S_{MIN}, S_{DIFF}$ can also be computed on the masked arrays :

```
In [70]: plt.imshow(Rmasked.mask[0],cmap='gray')
Out[70]: <matplotlib.image.AxesImage at 0x127cb1390>
```



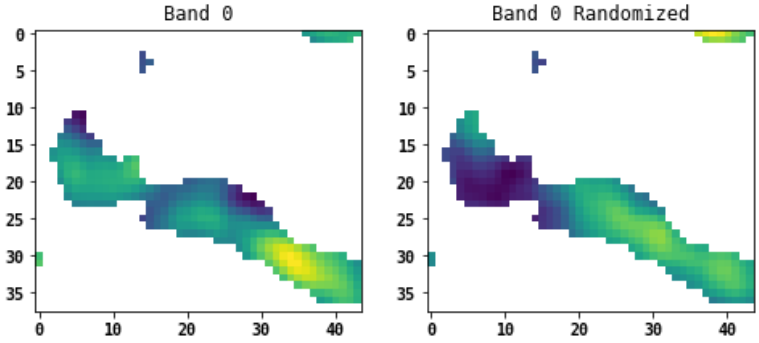

```
In [71]: Smasked = np.ma.cumsum(Rmasked,axis=0)
Smasked_max= np.ma.max(Smasked,axis=0)
Smasked_min= np.ma.min(Smasked,axis=0)
Smasked_diff=Smasked_max-Smasked_min
fig,ax=plt.subplots(1,3,figsize=(16,4))
vmin=Smasked_min.min()
vmax=Smasked_max.max()
p=ax[0].imshow(Smasked_max,vmin=vmin,vmax=vmax)
ax[0].set_title('$S_{max}$')
ax[1].imshow(Smasked_min,vmin=vmin,vmax=vmax)
ax[1].set_title('$S_{min}$')
ax[2].imshow(Smasked_diff,vmin=vmin,vmax=vmax)
ax[2].set_title('$S_{diff}$')
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
_=fig.colorbar(p,cax=cbar_ax)
```



Bootstrapping over the masked change point candidates

We can now perform the bootstrapping analysis over the not masked out values. For efficient computing we permutate the index of the time axis.

In [72]:
```
random_index=np.random.permutation(Rmasked.shape[0])
Rrandom=Rmasked[random_index,:,:]

fig,ax=plt.subplots(1,2,figsize=(8,4))
ax[0].imshow(Rmasked[0])
ax[0].set_title('Band 0')
ax[1].imshow(Rrandom[0])
_=ax[1].set_title('Band 0 Randomized')
```



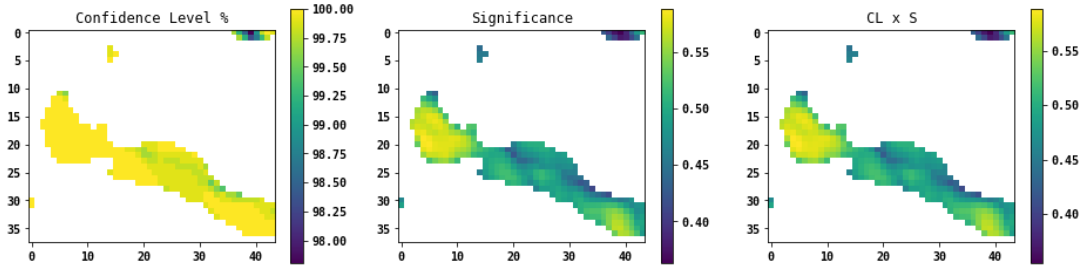
In [73]:
```
Smasked_max=np.ma.max(Smasked,axis=0)
```

Below is the numpy based implementation of the bootstrapping over all pixels. Note the efficient implementation using nympy masked arrays.

In [74]:
```
n_bootstraps=1000  # bootstrap sample size

# to keep track of the maxium Sdiff of the bootstrapped sample:
Sdiff_random_max = np.ma.copy(Smasked_diff)
Sdiff_random_max[~Sdiff_random_max.mask]=0
# to compute the Sdiff sums of the bootstrapped sample:
Sdiff_random_sum = np.ma.copy(Smasked_diff)
Sdiff_random_sum[~Sdiff_random_max.mask]=0
# to keep track of the count of the bootstrapped sample
n_Sdiff_gt_Sdiff_random = np.ma.copy(Smasked_diff)
n_Sdiff_gt_Sdiff_random[~n_Sdiff_gt_Sdiff_random.mask]=0
for i in range(n_bootstraps):
    # For efficiency, we shuffle the time axis index and use that
    #to randomize the masked array
    random_index=np.random.permutation(Rmasked.shape[0])
    # Randomize the time step of the residuals
    Rrandom = Rmasked[random_index,:,:]
    Srandom = np.ma.cumsum(Rrandom,axis=0)
    Srandom_max=np.ma.max(Srandom,axis=0)
    Srandom_min=np.ma.min(Srandom,axis=0)
    Sdiff_random=Srandom_max-Srandom_min
    Sdiff_random_sum += Sdiff_random
    Sdiff_random_max[np.ma.greater(Sdiff_random,Sdiff_random_max)]=\
    Sdiff_random[np.ma.greater(Sdiff_random,Sdiff_random_max)]
    n_Sdiff_gt_Sdiff_random[np.ma.greater(Smasked_diff,Sdiff_random)] += 1
```

Now we can compute for all pixels the confidence level CL, the change point significance metric $CP_{significance} and the product of the two as our confidence metric for identified changepoints.

In [75]:
```
CL = n_Sdiff_gt_Sdiff_random/n_bootstraps
CP_significance = 1.- (Sdiff_random_sum/n_bootstraps)/Sdiff
#Plot
fig,ax=plt.subplots(1,3,figsize=(16,4))
a = ax[0].imshow(CL*100)
fig.colorbar(a,ax=ax[0])
ax[0].set_title('Confidence Level %')
a = ax[1].imshow(CP_significance)
fig.colorbar(a,ax=ax[1])
ax[1].set_title('Significance')
a = ax[2].imshow(CL*CP_significance)
fig.colorbar(a,ax=ax[2])
_=ax[2].set_title('CL x S')
```

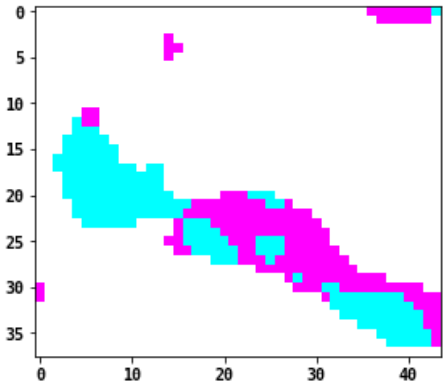


Now if we were to set a threshold of 0.5 for the product as identified change our change map would look like the following figure:

In [76]:
```
cp_thres=0.5
```

In [77]:
```
plt.imshow(CL*CP_significance <  cp_thres,cmap='cool')
```

Out[77]:
```
<matplotlib.image.AxesImage at 0x126bf49e8>
```



Our last step is the idenficiaton of the change points is to extract the timing of the change. We will produce a raster layer that shows the band number of this first date after detected change. We will make use of the numpy indexing scheme. First, we create a combined mask of the first threshold and the identified change points after the bootstrapping. For this we use the numpy "mask_or" operation.

In [78]:
```
# make a mask of our change points from the new threshold and the previous mask
cp_mask=np.ma.mask_or(CL*CP_significance<cp_thres,CL.mask)
# Broadcast the mask to the shape of the masked S curves
cp_mask2 = np.broadcast_to(cp_mask,Smasked.shape)
# Make a numpy masked array with this mask
CPraster = np.ma.array(Smasked.data,mask=cp_mask2)
```

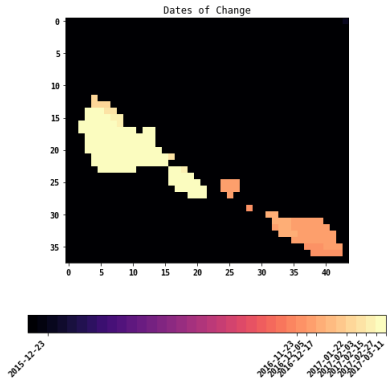
To retrieve the dates of the change points we find the band indices in the time series along the time axis where the the maximum of the cumulative sums was located. Numpy offers the "argmax" function for this purpose.

In [79]:
```
CP_index= np.ma.argmax(CPraster,axis=0)
change_indices = list(np.unique(CP_index))
change_indices.remove(0)
print(change_indices)
# Look up the dates from the indices to get the change dates
alldates=tindex[tindex>'2015-10-31']
change_dates=[str(alldates[x+1].date()) for x in change_indices]
print(change_dates)
```

```
[2, 27, 28, 29, 32, 33, 34, 35, 36]
['2015-12-23', '2016-11-23', '2016-12-05', '2016-12-17', '2017-01-22', '2017-02-
03', '2017-02-15', '2017-02-27', '2017-03-11']
```

Lastly, we visualize the change dates by showing the CP_index raster and label the change dates.

In [80]:
```
ticks=change_indices
ticklabels=change_dates

cmap=plt.cm.get_cmap('magma',ticks[-1])
fig, ax = plt.subplots(figsize=(8,8))
cax = ax.imshow(CP_index,interpolation='nearest',cmap=cmap)
# fig.subplots_adjust(right=0.8)
# cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
# fig.colorbar(p,cax=cbar_ax)

ax.set_title('Dates of Change')
# cbar = fig.colorbar(cax,ticks=ticks)
cbar=fig.colorbar(cax,ticks=ticks,orientation='horizontal')
_=cbar.ax.set_xticklabels(ticklabels,size=10,rotation=45,ha='right')
```



Secondary Change Points

After detection of a change point in the time series we can split the series in before and after change subsets. For forest degradation or deforestation detection for example this could apply when over the course of a multi-year time series selective logging precedes a clearing event or conversion of a logged plot to agriculture or regrowth, which show typically different time series profiles of radar backscatter. The approach to detect secondary change points would be to repeat analysis of the time series split into before and after change point detection.

Conclusion

Pandas and numpy are powerful open source scripting tools to implement change point detection on large data stacks. For image based analysis numpy offers more efficient implementations compared to pandas, whereas pandas is more powerful in date time processing, e.g. time-weighted interpolation.

Solutions

In [ ]:
```
# 1.
ts[ts<-11].index
```

In [ ]:
```
# 2.
gradient_lag1 = ts.diff(1)
gradient_lag1.plot()
```

In [ ]:
```
# 3.
gradient_lag1.min()
```

In [ ]:
```
# 4.
gradient_lag1[gradient_lag1==gradient_lag1.min()]
```

In [ ]:
```
before = gradient_lag1[gradient_lag1==gradient_lag1.min()].index[0]
before
```

In [ ]:
```
after=tindex[tindex>before][0]
after
```

SAR Training Workshop for Forest Applications

PART 5 - SAR/Optical (NDVI) Time Series Analysis

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2019

In this chapter we compare time series data of C-band Backscatter and Landsat 8 Normalized Difference Vegetation Index (NDVI) over a forested site in Southern Niger.

In [1]:
```
# Importing relevant python packages
import pandas as pd
import gdal
import numpy as np
import time,os
from skimage import exposure # to enhance image display

# For plotting
%matplotlib inline
import matplotlib.pylab as plt
import matplotlib.patches as patches
import matplotlib.cm as cm

font = {'family' : 'monospace',
          'weight' : 'bold',
          'size'   : 18}
plt.rc('font',**font)

# Define a helper function for a 4 part figure with backscatter, NDVI and False
 Color Infrared
def ebd_plot(bandnbrs):
    fig,ax=plt.subplots(2,2,figsize=(16,16))
    # Bands for sentinel and landsat:
    # Sentinel VV
    sentinel_vv=img_handle[0].GetRasterBand(bandnbrs[0]).ReadAsArray(*subset_sen
tinel)
    sentinel_vv=20.*np.log10(sentinel_vv)-83 # Covert to dB
    # Sentinel VH
    sentinel_vh=img_handle[1].GetRasterBand(bandnbrs[0]).ReadAsArray(*subset_sen
tinel)
    sentinel_vh=20.*np.log10(sentinel_vh)-83 # Covert to dB
    # # Landsat False Color InfraRed
    r=img_handle[5].GetRasterBand(bandnbrs[1]).ReadAsArray(*subset_landsat)/1000
0.
    g=img_handle[4].GetRasterBand(bandnbrs[1]).ReadAsArray(*subset_landsat)/1000
0.
    b=img_handle[3].GetRasterBand(bandnbrs[1]).ReadAsArray(*subset_landsat)/1000
0.
    fcir=np.dstack((r,g,b))
    for i in range(fcir.shape[2]):
        fcir[:,:,i] = exposure.\
        equalize_hist(fcir[:,:,i],
        mask=~np.equal(fcir[:,:,i],-.9999))
```

```
    # Landsat NDVI
    landsat_ndvi=img_handle[2].GetRasterBand(bandnbrs[1]).ReadAsArray(*subset_la
ndsat)
    mask=landsat_ndvi==-9999
    landsat_ndvi = landsat_ndvi/10000. # Scale to real NDVI value
    landsat_ndvi[mask]=np.nan
    svv = ax[0][0].imshow(sentinel_vv,cmap='jet',vmin=np.nanpercentile(sentinel_
vv,5),
                      vmax=np.nanpercentile(sentinel_vv,95))
    cb = fig.colorbar(svv,ax=ax[0][0],orientation='horizontal')
    cb.ax.set_title('C-VV $\gamma^o$ [dB]')
    svh = ax[0][1].imshow(sentinel_vh,cmap='jet',vmin=np.nanpercentile(sentinel_
vh,5),
                      vmax=np.nanpercentile(sentinel_vh,95))
    cb = fig.colorbar(svh,ax=ax[0][1],orientation='horizontal')
    cb.ax.set_title('C-VH $\gamma^o$ [dB]')

    nvmin=np.nanpercentile(landsat_ndvi,5)
    nvmax=np.nanpercentile(landsat_ndvi,95)
```

```
    # nvmin=-1
    # nvmax=1
    nax = ax[1][0].imshow(landsat_ndvi,cmap='jet',vmin=nvmin,
                      vmax=nvmax)
    cb = fig.colorbar(nax,ax=ax[1][0],orientation='horizontal')
    cb.ax.set_title('NDVI')

    fc= ax[1][1].imshow(fcir)
    # cb = fig.colorbar(fc,cmap=cm.gray,ax=ax[1][1],orientation='horizontal')
    # cb.ax.set_title('False Color Infrared')

    ax[0][0].axis('off')
    ax[0][1].axis('off')
    ax[1][0].axis('off')
    ax[1][1].axis('off')
    ax[0][0].set_title('Sentinel-1 C-VV {}'.format(stindex[bandnbrs[0]-1].date
()))
    ax[0][1].set_title('Sentinel-1 C-VH {}'.format(stindex[bandnbrs[0]-1].date
()))
    ax[1][0].set_title('Landsat-8 NDVI {}'.format(ltindex[bandnbrs[1]-1].date
()))
    ax[1][1].set_title('Landsat-8 False Color IR {}'.format(ltindex[bandnbrs[1]-
1].date()))
    _=fig.suptitle('Sentinel-1 Backscatter and Landsat NDVI and FC IR',size=16)
```

Set Project Directory and Filenames

West Africa - Biomass Site

In [2]:
```
datadirectory='C:\\Users\\loaner.SERVIRLOAN-5057.001\\Downloads\\BIOsS1'
#datadirectory='/dev/shm/projects/c401servir/wa/BIOsS1'
sentinel1_datefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
sentinel1_imagefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
sentinel1_imagefile_cross='S32631X398020Y1315440sS1_A_vh_0001_mtfil.vrt'
landsat8_ndvi='landsat/L8_192_052_NDVI.vrt'
landsat8_b3='landsat/L8_192_052_B3.vrt'
landsat8_b4='landsat/L8_192_052_B4.vrt'
landsat8_b5='landsat/L8_192_052_B5.vrt'
landsat8_datefile='landsat/L8_192_052_NDVI.dates'
```

In [3]:
```
# Switch to the data directory
os.chdir(os.path.join(datadirectory))
```

Acquisition Dates

Read from the Dates file the dates in the time series and make a pandas date index

In [4]:
```
sdates=open(sentinel1_datefile).readlines()
stindex=pd.DatetimeIndex(sdates)
j=1
print('Bands and dates for',sentinel1_imagefile)
for i in stindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt
   1 2015-03-22    2 2015-04-03    3 2015-04-15    4 2015-05-09    5 2015-05-21
   6 2015-06-02    7 2015-06-14    8 2015-06-26    9 2015-07-08   10 2015-07-20
  11 2015-08-01   12 2015-08-13   13 2015-08-25   14 2015-09-06   15 2015-09-18
  16 2015-09-30   17 2015-10-12   18 2015-10-24   19 2015-11-17   20 2015-11-29
  21 2015-12-11   22 2015-12-23   23 2016-01-04   24 2016-01-28   25 2016-02-09
  26 2016-03-04   27 2016-03-16   28 2016-03-28   29 2016-04-09   30 2016-04-21
  31 2016-05-03   32 2016-05-15   33 2016-05-27   34 2016-06-08   35 2016-07-02
  36 2016-07-14   37 2016-07-26   38 2016-08-07   39 2016-08-19   40 2016-08-31
  41 2016-09-12   42 2016-09-24   43 2016-10-06   44 2016-10-18   45 2016-10-30
  46 2016-11-11   47 2016-11-23   48 2016-12-05   49 2016-12-17   50 2016-12-29
  51 2017-01-10   52 2017-01-22   53 2017-02-03   54 2017-02-15   55 2017-02-27
  56 2017-03-11   57 2017-03-23   58 2017-04-04   59 2017-04-16   60 2017-04-28
  61 2017-05-10   62 2017-05-22   63 2017-06-03   64 2017-06-15   65 2017-06-27
  66 2017-07-09   67 2017-07-21   68 2017-08-02   69 2017-08-14   70 2017-08-26
  71 2017-09-07   72 2017-09-19   73 2017-10-13   74 2017-10-25   75 2017-11-06
  76 2017-11-18   77 2017-11-30
```

In [5]:
```
ldates=open(landsat8_datefile).readlines()
ltindex=pd.DatetimeIndex(ldates)
j=1
print('Bands and dates for',landsat8_ndvi)
for i in ltindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for landsat/L8_192_052_NDVI.vrt
   1 2015-01-13    2 2015-01-29    3 2015-02-14    4 2015-03-18    5 2015-04-03
   6 2015-04-19    7 2015-05-05    8 2015-05-21    9 2015-06-06   10 2015-06-22
  11 2015-07-08   12 2015-07-24   13 2015-08-09   14 2015-08-25   15 2015-09-10
  16 2015-09-26   17 2015-10-12   18 2015-10-28   19 2015-11-13   20 2015-11-29
  21 2015-12-15   22 2015-12-31   23 2016-01-16   24 2016-02-01   25 2016-02-17
  26 2016-03-04   27 2016-03-20   28 2016-04-05   29 2016-04-21   30 2016-05-07
  31 2016-05-23   32 2016-06-08   33 2016-06-24   34 2016-07-10   35 2016-07-26
  36 2016-08-11   37 2016-08-27   38 2016-09-12   39 2016-09-28   40 2016-10-14
  41 2016-10-30   42 2016-11-15   43 2016-12-01   44 2016-12-17   45 2017-01-02
  46 2017-01-18   47 2017-02-03   48 2017-02-19   49 2017-03-07   50 2017-03-23
  51 2017-04-08   52 2017-04-24
```

Projection and Georeferencing Information of the SAR and Optical Time Series Data Stacks

For processing of the imagery in this notebook we generate a list of image handles and retrieve projection and georeferencing information. We print out the retrieved information.

In [6]:
```
imagelist=[sentinel1_imagefile,sentinel1_imagefile_cross,landsat8_ndvi,landsat8_
b3,landsat8_b4,landsat8_b5]
geotrans=[]
proj=[]
img_handle=[]
xsize=[]
ysize=[]
bands=[]
for i in imagelist:
    img_handle.append(gdal.Open(i))
    geotrans.append(img_handle[-1].GetGeoTransform())
    proj.append(img_handle[-1].GetProjection())
    xsize.append(img_handle[-1].RasterXSize)
    ysize.append(img_handle[-1].RasterYSize)
    bands.append(img_handle[-1].RasterCount)
# for i in proj:
#     print(i)
# for i in geotrans:
#     print(i)
# for i in zip(['C-VV','C-VH','NDVI','B3','B4','B5'],bands,ysize,xsize):
#     print(i)
```

Display SAR and NDVI Images

First, depending on the capacity of the computer we might want to define a subset. We will choose the subset in the raster extension of the Sentinel-1 Image and use the geotransformation information to extract the corresponding subset in the Landsat Image. We assume that the images have the same upper left coordinate. The we can compute the offsets and extent in the Landsat image as follows:

$$x_{cal} = \frac{xres_{sentinel-1}}{xres_{landsat}}$$

$$y_{cal} = \frac{yres_{sentinel-1}}{yres_{landsat}}$$

We can use these calibration factors to get the landsat subset as follows:

- $xoff_{landsat} = xoff_{sentinel-1} \times x_{cal}$
- $yoff_{landsat} = yoff_{sentinel-1} \times y_{cal}$
- $xsize_{landsat} = xsize_{sentinel-1} \times x_{cal}$
- $ysize_{landsat} = ysize_{sentinel-1} \times y_{cal}$

(xoffset,yoffset,xsize,ysize)

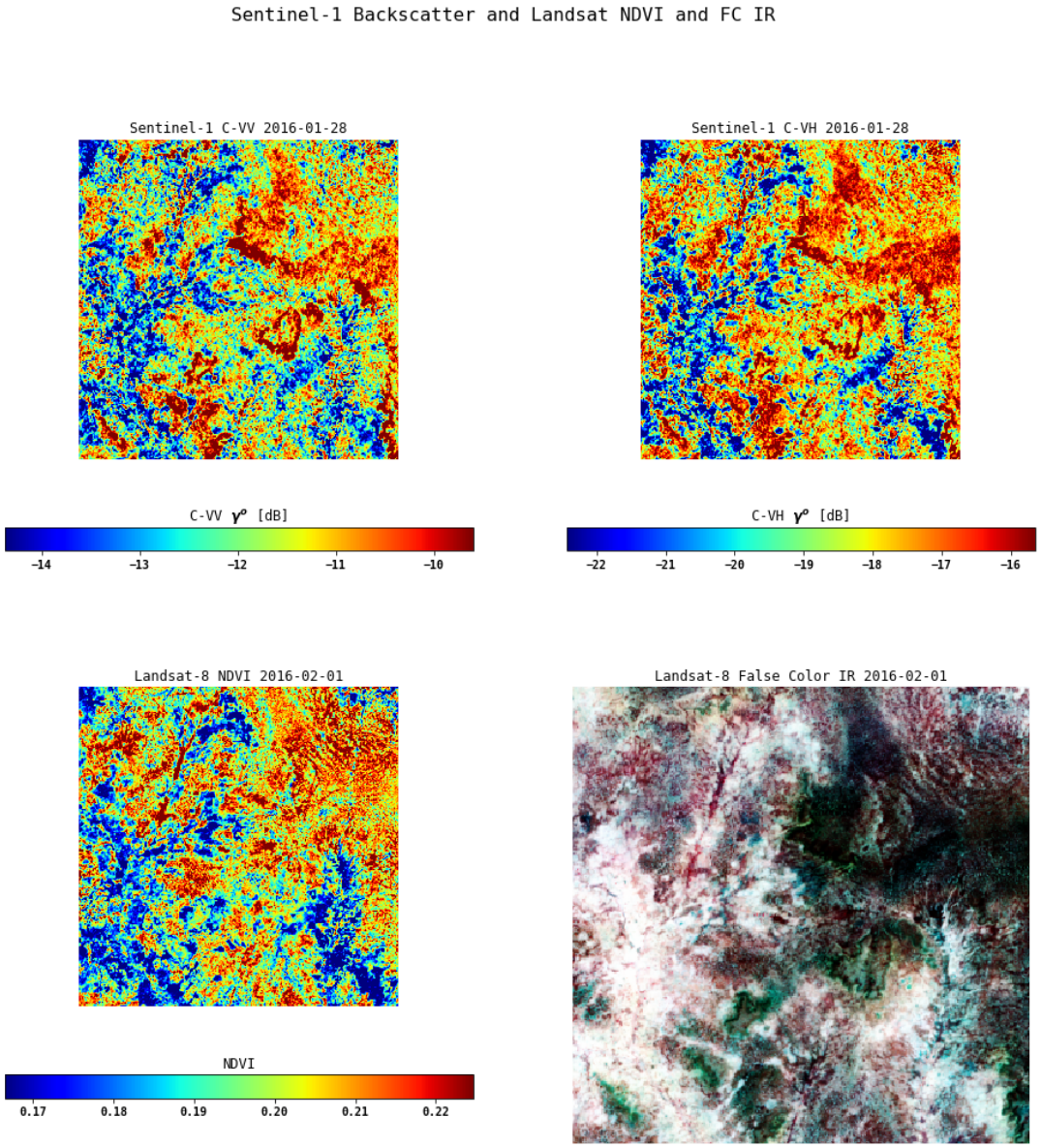
In [7]:
```
subset_sentinel=None
subset_sentinel=(570,40,500,500)  # Adjust or comment out if you don't want a su
bset
if subset_sentinel == None:
    subset_sentinel=(0,0,img_handle[0].RasterXSize,img_handle[0].RasterYSize)
    subset_landsat=(0,0,img_handle[2].RasterXSize,img_handle[2].RasterYSize)
else:
    xoff,yoff,xsize,ysize=subset_sentinel
    xcal=geotrans[0][1]/geotrans[2][1]
    ycal=geotrans[0][5]/geotrans[2][5]
    subset_landsat=(int(xoff*xcal),int(yoff*ycal),int(xsize*xcal),int(ysize*ycal
))

print('Subset Sentinel-1',subset_sentinel,'\nSubset Landsat   ',subset_landsat)
```

```
Subset Sentinel-1 (570, 40, 500, 500)
Subset Landsat    (380, 26, 333, 333)
```

Now we can pick the bands and plot the Sentinel-1 and Landsat NDVI images of the subset. Change the band numbers to the bands we are interested in.

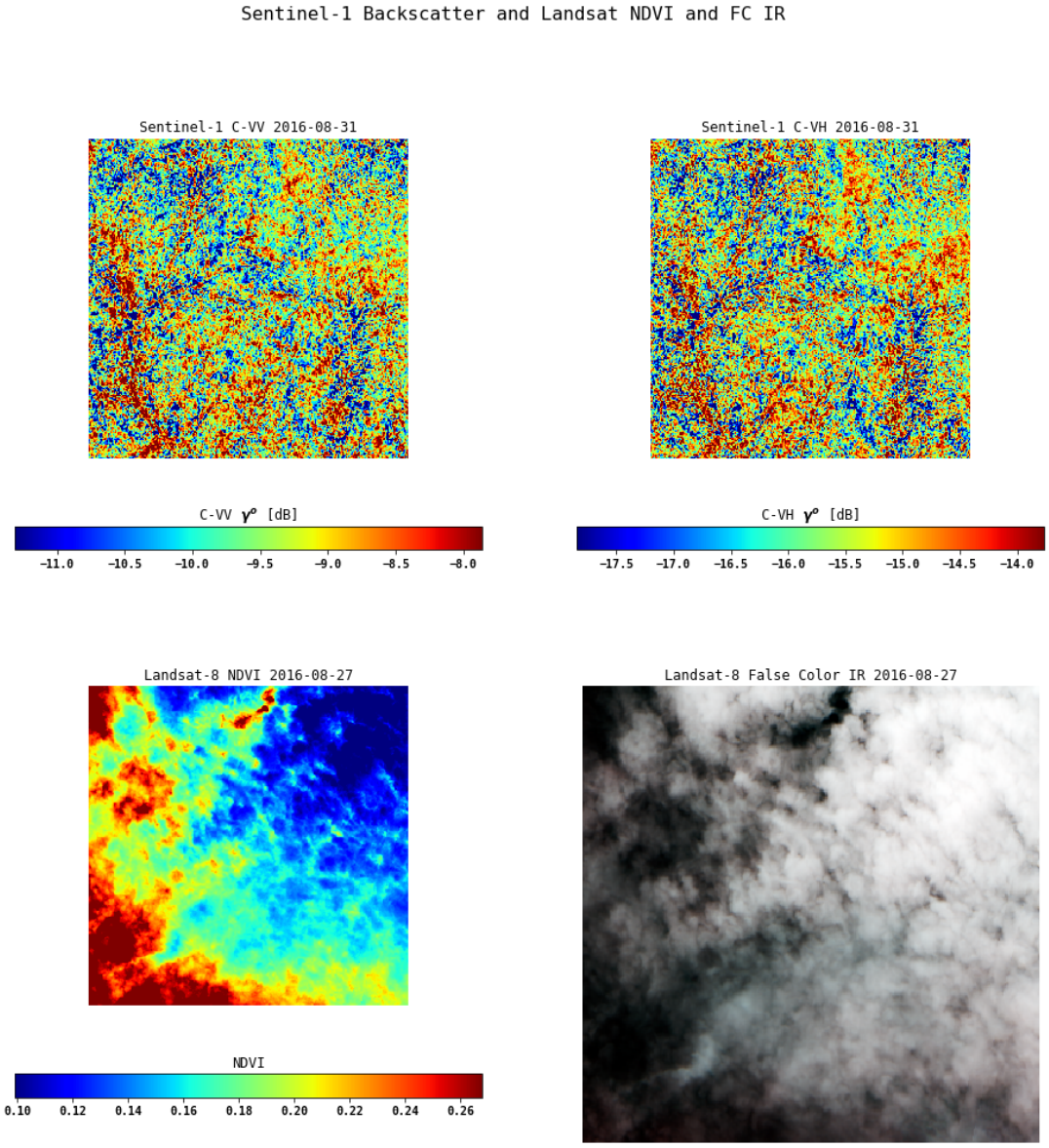
Dry Season Plot

In [8]:
```
# Dry season plot
bandnbrs=(24,24)
ebd_plot(bandnbrs)
```



Wet Season Plot

In [9]:
```
# Wet season plot
bandnbrs=(40,37)
ebd_plot(bandnbrs)
```



In the figure above, for band 24 of Sentinel-1 and 24 of NDVI, which was acquired three days after the Sentinel-1 image, there is an inverse relationship. Where Sentinel-1 exhibits low backscatter, NDVI shows relatively higher NDVI. What are the reasons for this in this environment?

Exercise

Pick different bands to compare. Look at the list of the dates for SAR data and Landsat data acquisitions in the above. One good option is to compare bands from the dry and wet seasons 2016.

Time Series Profiles of C-Band Backscatter and NDVI

We compute the image means of each time step in the time series stack and plot them together.

Prepare Sentinel-1 and NDVI Data stacks

Sentinel time series stack

In [10]:
```
caldB=-83
calPwr = np.power(10.,caldB/10.)

s_ts=[]
for idx in (0,1):
    means=[]
    for i in range(bands[idx]):
        rs=img_handle[idx].GetRasterBand(i+1).ReadAsArray(*subset_sentinel)
        # 1. Conversion to Power
        rs_pwr=np.power(rs,2.)*calPwr
        rs_means_pwr = np.mean(rs_pwr)
        rs_means_dB = 10.*np.log10(rs_means_pwr)
        means.append(rs_means_dB)
    s_ts.append(pd.Series(means,index=stindex))
```

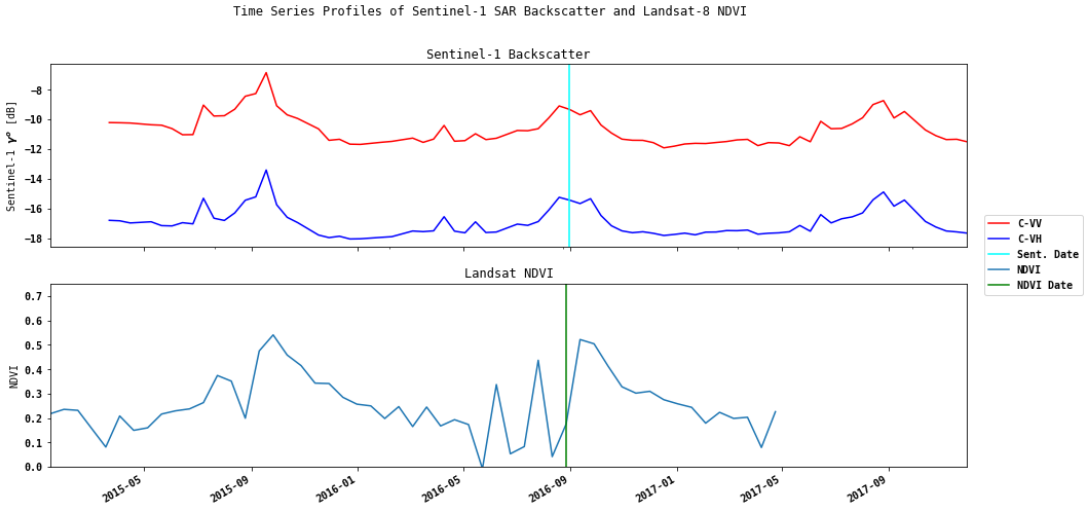
Landsat NDVI time series stack

In [11]:
```
means=[]
idx=2
for i in range(bands[idx]):
    r=img_handle[idx].GetRasterBand(i+1).ReadAsArray(*subset_landsat)
    means.append(r[r!=-9999].mean()/10000.)
l_ts=pd.Series(means,index=ltindex)
```

Joint Plot of SAR Backscatter and NDVI of Image Subset Means

Now we plot the time series of the SAR backscatter and NDVI values scaled to the same time axis. We also show the time stamps for the images we display above.

In [12]:
```
fig, ax = plt.subplots(2,1,figsize=(16,8))
# ax1.plot(s_ts.index,s_ts.values, 'r-')
s_ts[0].plot(ax=ax[0],color='red',label='C-VV',xlim=(min(min(ltindex),min(stinde
x)),
                                                    max(max(ltindex),max(stinde
x))))
s_ts[1].plot(ax=ax[0],color='blue',label='C-VH')
ax[0].set_xlabel('Date')
ax[0].set_ylabel('Sentinel-1 $\gamma^o$ [dB]')

# Make the y-axis label, ticks and tick labels match the line color. ax1.set_yla
bel('exp', color='b')
# ax1.tick_params('y', colors='b')
# ax[1] = ax1.twinx()
# s_ts.plot(ax=ax[1],share=ax[0])
l_ts.plot(ax=ax[1],sharex=ax[0],label='NDVI',xlim=(min(min(ltindex),min(stindex
)),
                                                    max(max(ltindex),max(stinde
x))),ylim=(0,0.75))
# ax[1].plot(l_ts.index,l_ts.values,color='green',label='NDVI')
ax[1].set_ylabel('NDVI')
ax[0].set_title('Sentinel-1 Backscatter')
ax[1].set_title('Landsat NDVI')

ax[0].axvline(stindex[bandnbrs[0]-1],color='cyan',label='Sent. Date')
ax[1].axvline(ltindex[bandnbrs[1]-1],color='green',label='NDVI Date')
_=fig.legend(loc='center right')
_=fig.suptitle('Time Series Profiles of Sentinel-1 SAR Backscatter and Landsat-8
NDVI ')
# fig.tight_layout()
```



Comparison of time series profiles at point locations.

We will pick a pixel location in the SAR image, find the corresponding location in the Landsat NDVI stack and plot the joint time series.

First let's pick a pixel location in the SAR image (i.e. the reference image)

We use the geotrans info to find the same location in the Landsat image

In [13]:
```
sarloc=(2000,2000)
ref_x=geotrans[0][0]+sarloc[0]*geotrans[0][1]
ref_y=geotrans[0][3]+sarloc[1]*geotrans[0][5]
print('UTM Coordinates      ',ref_x,ref_y)
print('SAR pixel/line       ',sarloc[0],sarloc[1])
target_pixel=round((ref_x-geotrans[2][0])/geotrans[2][1])
target_line=round((ref_y-geotrans[2][3])/geotrans[2][5])
print('Landsat pixel/line   ',target_pixel,target_line)
```

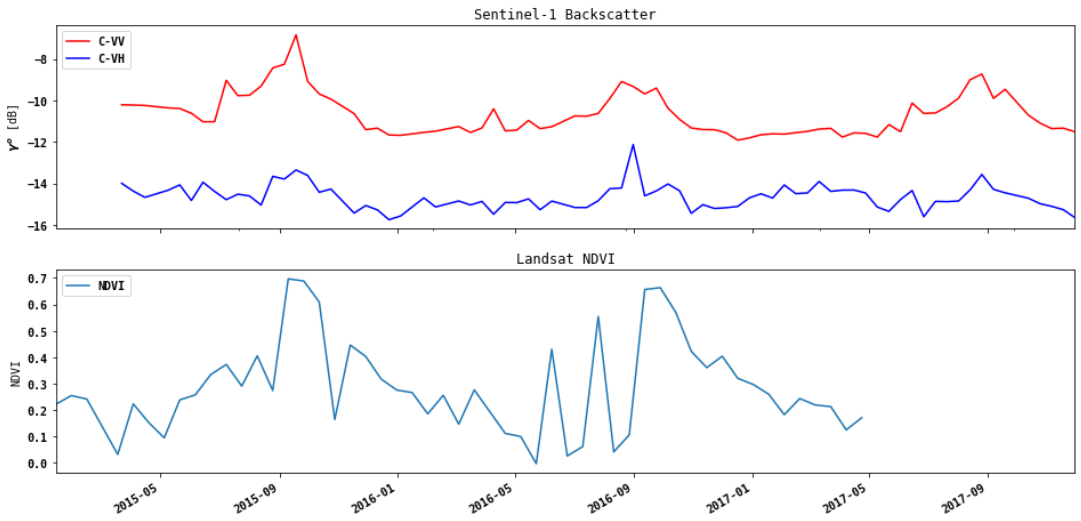
```
UTM Coordinates       438020.0 1350960.0
SAR pixel/line        2000 2000
Landsat pixel/line    1333 1334
```

Read the image data at these locations

In [14]:
```
s_ts_pixel=[]
for idx in (0,1):
    means=[]
    for i in range(bands[idx]):
        rs=img_handle[idx].GetRasterBand(i+1).ReadAsArray(*sarloc,6,6)
        # 1. Conversion to Power
        rs_pwr=np.power(rs,2.)*calPwr
        rs_means_pwr = np.mean(rs_pwr)
        rs_means_dB = 10.*np.log10(rs_means_pwr)
        means.append(rs_means_dB)
    s_ts_pixel.append(pd.Series(means,index=stindex))

means=[]
idx=2
for i in range(bands[idx]):
    r=img_handle[idx].GetRasterBand(i+1).ReadAsArray(target_pixel,target_line,4,
4)
    means.append(np.nanmean(r)/10000.)
l_ts_pixel=pd.Series(means,index=ltindex)
```

Plot the joint time series.

In [15]:
```
fig, ax = plt.subplots(2,1,figsize=(16,8))
# ax1.plot(s_ts.index,s_ts.values, 'r-')
s_ts[0].plot(ax=ax[0],color='red',label='C-VV',xlim=(min(min(ltindex),min(stinde
x)),
                                                    max(max(ltindex),max(stinde
x))))
s_ts_pixel[1].plot(ax=ax[0],color='blue',label='C-VH')
ax[0].set_xlabel('Date')
ax[0].set_ylabel('$\gamma^o$ [dB]')

# Make the y-axis label, ticks and tick labels match the line color. ax1.set_yla
bel('exp', color='b')
# ax1.tick_params('y', colors='b')
# ax[1] = ax1.twinx()
# s_ts.plot(ax=ax[1],share=ax[0])
l_ts_pixel.plot(ax=ax[1],sharex=ax[0],label='NDVI',xlim=(min(min(ltindex),min(st
index)),
                                                    max(max(ltindex),max(stinde
x))))
# ax[1].plot(l_ts.index,l_ts.values,color='green',label='NDVI')
ax[1].set_ylabel('NDVI')
ax[0].set_title('Sentinel-1 Backscatter')
ax[1].set_title('Landsat NDVI')
_=ax[0].legend(loc='upper left')
_=ax[1].legend(loc='upper left')
# fig.tight_layout()
```



Interpret these time series profiles. While generally the seasonal trends are visible in both like (VV) and cross-polarized (VH) data, and correlate well with the NDVI temporal profile, the cross-polarized response is less pronounced at the example pixel location UTM Coordinates Zone 31N 438020.0 1350960.0.

EXERCISE

Pick different pixel locations and replot the figure above. Interpret the result with respect to forest, non-forest, deforestation and forest degradation signatures. In your interpretation look for image signals of strong rain events in the SAR data and cloud covered scenes in the Landsat imagery.

SAR Training Workshop for Forest Applications
How to Make RGB Composites from Dual-Polarimetric Data

Josef Kellndorfer, Ph.D., President and Senior Scientist, Earth Big Data, LLC

Revision date: January 2019

In this chapter we introduce how to make a three band color composite and save it

Import Python modules

In [1]:
```
import os,sys,gdal
%matplotlib inline
import matplotlib.pylab as plt
import matplotlib.patches as patches  # Needed to draw rectangles
from skimage import exposure # to enhance image display
import numpy as np
import pandas as pd
```

In [2]:
```
# Select the project data set and time series data
```

In [3]:
```
# West Africa - Biomass Site
datapath='Users/rmuench/Downloads/wa/BIOsS1/'
datefile='S32631X398020Y1315440sS1_A_vv_0001_mtfil.dates'
imagefile_like='S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt'
imagefile_cross='S32631X398020Y1315440sS1_A_vh_0001_mtfil.vrt'
```

In [4]:
```
os.chdir(datapath)
```

We are defining two helper functions for this task

- **CreateGeoTiff()** to write out images
- **dualpol2rgb()** to compute various metrics from a time series data stack

In [5]:

```
def CreateGeoTiff(Name, Array, DataType, NDV,bandnames=None,ref_image=None,
                  GeoT=None, Projection=None):
    # If it's a 2D image we fake a third dimension:
    if len(Array.shape)==2:
        Array=np.array([Array])
    if ref_image==None and (GeoT==None or Projection==None):
        raise RuntimeWarning('ref_image or settings required.')
    if bandnames != None:
        if len(bandnames) != Array.shape[0]:
            raise RuntimeError('Need {} bandnames. {} given'
                               .format(Array.shape[0],len(bandnames)))
    else:
        bandnames=['Band {}'.format(i+1) for i in range(Array.shape[0])]
    if ref_image!= None:
        refimg=gdal.Open(ref_image)
        GeoT=refimg.GetGeoTransform()
        Projection=refimg.GetProjection()
    driver= gdal.GetDriverByName('GTIFF')
    Array[np.isnan(Array)] = NDV
    DataSet = driver.Create(Name,
            Array.shape[2], Array.shape[1], Array.shape[0], DataType)
    DataSet.SetGeoTransform(GeoT)
    DataSet.SetProjection( Projection)
    for i, image in enumerate(Array, 1):
        DataSet.GetRasterBand(i).WriteArray( image )
        DataSet.GetRasterBand(i).SetNoDataValue(NDV)
        DataSet.SetDescription(bandnames[i-1])
    DataSet.FlushCache()
    return Name
```

In [6]:

```
def dualpol2rgb(like,cross,sartype='amp',ndv=0):
    CF=np.power(10.,-8.3)
    if np.isnan(ndv):
        mask=np.isnana(cross)
    else:
        mask=np.equal(cross,ndv)

    l = np.ma.array(like,mask=mask,dtype=np.float32)
    c = np.ma.array(cross,mask=mask,dtype=np.float32)

    if sartype=='amp':
        l=np.ma.power(l,2.)*CF
        c=np.ma.power(l,2.)*CF
    elif sartype=='dB':
        l=np.ma.power(10.,l/10.)
        c=np.ma.power(10.,c/10.)
    elif sartype=='pwr':
        pass
    else:
        print('invalid type ',sartype)
        raise RuntimeError

    if sartype=='amp':
        ratio=np.ma.sqrt(l/c)/10
        ratio[np.isinf(ratio.data)]=0.00001
    elif sartype=='dB':
        ratio=10.*np.ma.log10(l/c)
    else:
        ratio=l/c

    ratio=ratio.filled(ndv)

    rgb=np.dstack((like,cross,ratio.data))


    bandnames=('Like','Cross','Ratio')
    return rgb,bandnames,sartype


def any2amp(raster,sartype='amp',ndv=0):
    CF=np.power(10.,-8.3)
    mask=raster==ndv

    if sartype=='pwr':
        raster=np.sqrt(raster/CF)
    elif sartype=='dB':
        raster=np.ma.power(10.,(raster+83)/20.)
    elif sartype=='amp':
        pass
    else:
        print('invalid type ',sartype)
        raise RuntimeError

    raster[raster<1]=1
    raster[raster>65535]=65535
    raster[mask]=0
    raster=np.ndarray.astype(raster,dtype=np.uint16)
    return raster
```

Set the Dates

In [7]:
```
# Get the date indices via pandas
dates=open(datefile).readlines()
tindex=pd.DatetimeIndex(dates)
j=1
print('Bands and dates for',imagefile_like)
for i in tindex:
    print("{:4d} {}".format(j, i.date()),end=' ')
    j+=1
    if j%5==1: print()
```

```
Bands and dates for S32631X398020Y1315440sS1_A_vv_0001_mtfil.vrt
   1 2015-03-22    2 2015-04-03    3 2015-04-15    4 2015-05-09    5 2015-05-21
   6 2015-06-02    7 2015-06-14    8 2015-06-26    9 2015-07-08   10 2015-07-20
  11 2015-08-01   12 2015-08-13   13 2015-08-25   14 2015-09-06   15 2015-09-18
  16 2015-09-30   17 2015-10-12   18 2015-10-24   19 2015-11-17   20 2015-11-29
  21 2015-12-11   22 2015-12-23   23 2016-01-04   24 2016-01-28   25 2016-02-09
  26 2016-03-04   27 2016-03-16   28 2016-03-28   29 2016-04-09   30 2016-04-21
  31 2016-05-03   32 2016-05-15   33 2016-05-27   34 2016-06-08   35 2016-07-02
  36 2016-07-14   37 2016-07-26   38 2016-08-07   39 2016-08-19   40 2016-08-31
  41 2016-09-12   42 2016-09-24   43 2016-10-06   44 2016-10-18   45 2016-10-30
  46 2016-11-11   47 2016-11-23   48 2016-12-05   49 2016-12-17   50 2016-12-29
  51 2017-01-10   52 2017-01-22   53 2017-02-03   54 2017-02-15   55 2017-02-27
  56 2017-03-11   57 2017-03-23   58 2017-04-04   59 2017-04-16   60 2017-04-28
  61 2017-05-10   62 2017-05-22   63 2017-06-03   64 2017-06-15   65 2017-06-27
  66 2017-07-09   67 2017-07-21   68 2017-08-02   69 2017-08-14   70 2017-08-26
  71 2017-09-07   72 2017-09-19   73 2017-10-13   74 2017-10-25   75 2017-11-06
  76 2017-11-18   77 2017-11-30
```

In [8]:
```
# PICK A BAND NAUMBER
bandnbr=70
```

Open the image and get dimensions (bands,lines,pixels):

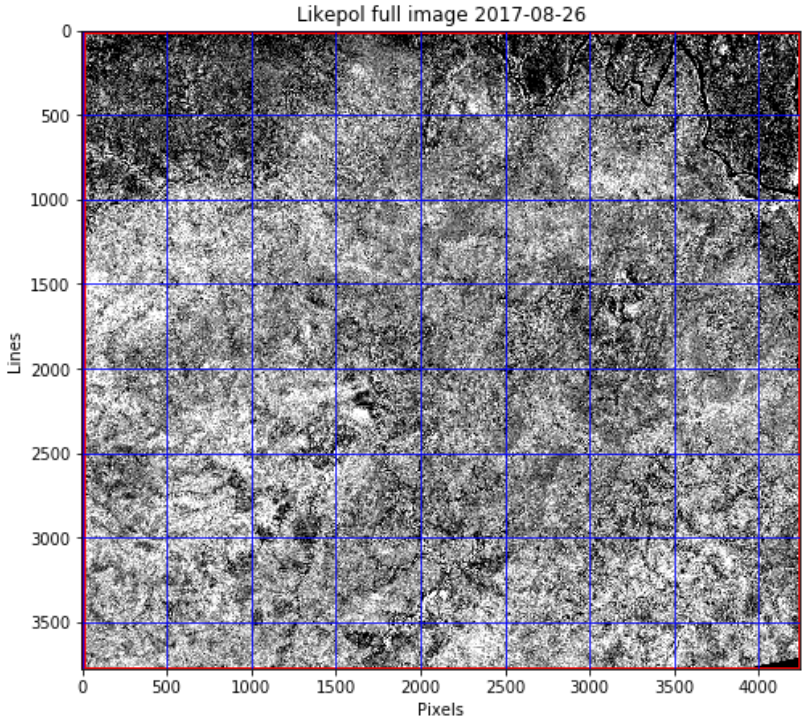
In [9]:
```
img_like=gdal.Open(imagefile_like)
img_cross=gdal.Open(imagefile_cross)
# Get Dimensions
print('Likepol ',img_like.RasterCount,img_like.RasterYSize,img_like.RasterXSize)
print('Crosspol',img_cross.RasterCount,img_cross.RasterYSize,img_cross.RasterXSiz
e)
```

```
Likepol  77 3776 4243
Crosspol 77 3776 4243
```

For a manageable size we can choose a 500x500 pixel subset to read the entire data stack (commented out). We also convert the amplitude data to power data right away and will perform the rest of the calculations on the power data to be mathmatically correct.

NOTE: Choose a different xsize/ysize in the subset if you need to.

In [10]:
```
subset=None
#subset=(3500,1000,500,500)   # (xoff,yoff,xsize,ysize)
if subset==None:
    subset=(0,0,img_like.RasterXSize,img_like.RasterYSize)

raster=img_like.GetRasterBand(bandnbr).ReadAsArray()
fig, ax = plt.subplots(figsize=(8,8))
ax.set_title('Likepol full image {}'
             .format(tindex[bandnbr-1].date()))
ax.imshow(raster,cmap='gray',vmin=np.nanpercentile(raster,5),vmax=np.nanpercentil
e(raster,95))
ax.grid(color='blue')
ax.set_xlabel('Pixels')
ax.set_ylabel('Lines')
# plot the subset as rectangle
if subset != None:
    _=ax.add_patch(patches.Rectangle((subset[0],subset[1]),
                                     subset[2],subset[3],
                                     fill=False,edgecolor='red',
                                     linewidth=3))
```



Make the RGB like/cross/ratio image

In [11]:
```
raster_like=img_like.GetRasterBand(bandnbr).ReadAsArray(*subset)
raster_cross=img_cross.GetRasterBand(bandnbr).ReadAsArray(*subset)
```

We make an RGB stack to display the like,cross, and ratio data as a color composite.

In [12]:
```
rgb,bandnames,sartype=dualpol2rgb(raster_like,raster_cross)
```

We are interested in displaying the image enhanced with histogram equalization.

We can use the function *exposure.equalize_hist()* from the skimage.exposure module

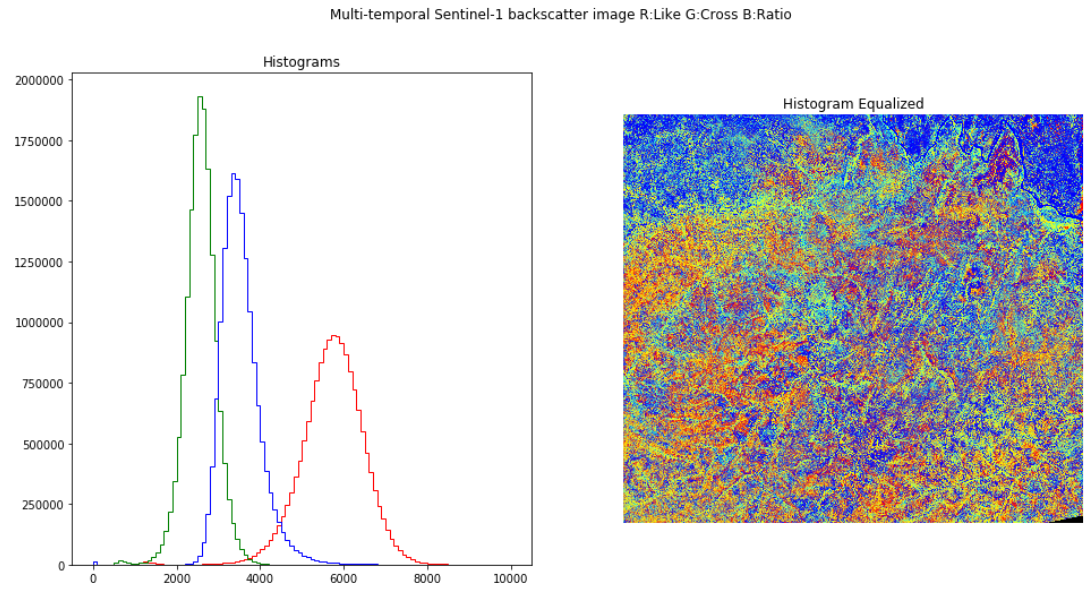
In [13]:
```
rgb_stretched=np.ndarray.astype(rgb.copy(),'float32')
# For each band we apply the strech
for i in range(rgb_stretched.shape[2]):
    rgb_stretched[:,:,i] = np.ndarray.astype(exposure.equalize_hist(rgb_stretched
[:,:,i],
    mask=~np.equal(rgb_stretched[:,:,i],0)),'float32')
```

In [14]:
```
rgb_stretched
```

Out[14]:
```
array([[[6.78325057e-01, 9.72052574e-01, 3.90828580e-01],
        [8.76864195e-01, 9.84529495e-01, 1.71240658e-01],
        [9.83614683e-01, 9.98086393e-01, 2.89953277e-02],
        ...,
        [5.28072240e-03, 1.07818116e-02, 9.95057702e-01],
        [6.99684722e-03, 1.37452455e-02, 9.93614554e-01],
        [3.84910442e-02, 9.95012820e-02, 9.67868745e-01]],

       [[5.00227690e-01, 9.16491270e-01, 5.65926731e-01],
        [6.34485126e-01, 9.05717731e-01, 4.34903175e-01],
        [8.29794526e-01, 9.68486011e-01, 2.25173086e-01],
        ...,
        [5.95453382e-03, 1.87972672e-02, 9.94502723e-01],
        [6.23923307e-03, 1.84695572e-02, 9.94265974e-01],
        [2.66998932e-02, 9.82193723e-02, 9.77528811e-01]],

       [[1.96703568e-01, 4.68630195e-01, 8.36585879e-01],
        [2.68557519e-01, 4.29826617e-01, 7.75374472e-01],
        [4.55813646e-01, 7.10059583e-01, 6.08429134e-01],
        ...,
        [1.46300010e-02, 1.38415113e-01, 9.87303555e-01],
        [1.15622561e-02, 9.69374627e-02, 9.89897549e-01],
        [2.56231278e-02, 1.70091271e-01, 9.78379309e-01]],

       ...,

       [[5.30907631e-01, 7.67807901e-01, 5.37106931e-01],
        [2.64292091e-01, 5.22130668e-01, 7.78782785e-01],
        [1.08426727e-01, 3.33118588e-01, 9.10773575e-01],
        ...,
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07]],

       [[4.51068103e-01, 8.14454377e-01, 6.13029897e-01],
        [2.54053503e-01, 6.20940566e-01, 7.87545741e-01],
        [1.29825607e-01, 3.66306335e-01, 8.92777562e-01],
        ...,
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07]],

       [[4.22746390e-01, 8.42523038e-01, 6.38080597e-01],
        [2.41145849e-01, 6.77982450e-01, 7.98704743e-01],
        [1.81992248e-01, 4.34495091e-01, 8.49451780e-01],
        ...,
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07],
        [1.24921982e-07, 1.87382980e-07, 2.49843964e-07]]], dtype=float32)
```

Now let's display the the histograms and equalized image side by side.

In [15]:
```
fig,ax = plt.subplots(1,2,figsize=(16,8))
fig.suptitle('Multi-temporal Sentinel-1 backscatter image R:{} G:{} B:{}'
             .format(bandnames[0],bandnames[1],bandnames[2]))
plt.axis('off')
ax[0].hist(rgb[:,:,0].flatten(),histtype='step',color='red',bins=100,range=(0,100
00))
ax[0].hist(rgb[:,:,1].flatten(),histtype='step',color='green',bins=100,range=(0,1
0000))
ax[0].hist(rgb[:,:,2].flatten(),histtype='step',color='blue',bins=100,range=(0,10
000))
ax[0].set_title('Histograms')
ax[1].imshow(rgb_stretched)
ax[1].set_title('Histogram Equalized')
_=ax[1].axis('off')
```



Write the images to an output file

Determine output geometry

First, we need to set the correct geotransformation and projection information. We retrieve the values from the input images and adjust by the subset:

In [16]:
```
proj=img_like.GetProjection()
geotrans=list(img_like.GetGeoTransform())

subset_xoff=geotrans[0]+subset[0]*geotrans[1]
subset_yoff=geotrans[3]+subset[1]*geotrans[5]
geotrans[0]=subset_xoff
geotrans[3]=subset_yoff
geotrans=tuple(geotrans)
geotrans
```

Out[16]:
```
(398020.0, 20.0, 0.0, 1390960.0, 0.0, -20.0)
```

Convert to 16bit Amplitude image

We use the root of the time series data stack name and append a _ts_metrics_.tif ending as filenames

Build a like/cross/ratio amplitude scaled GeoTIFF images

In [17]:
```
outbands=[]
for i in range(3):
    outbands.append(any2amp(rgb[:,:,i]))

imagename=imagefile_like.replace('_vv_','_lcr_').replace('.vrt','_{}.tif'.format(
dates[bandnbr-1].rstrip()))
bandnames=['Like','Cross','Ratio']
Array=np.array(outbands)
CreateGeoTiff(imagename,Array,gdal.GDT_UInt16,0,bandnames,GeoT=geotrans,Projectio
n=proj)
```

Out[17]:
```
'S32631X398020Y1315440sS1_A_lcr_0001_mtfil_20170826.tif'
```

This Image can now be loaded into QGIS or similar programs

Exercise

Change the bandnbr, generate a new rgb image and export it. Display in QGIS