In [1]:
```
# Import packages
import os
import glob

import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

import numpy as np
import rasterio
import xarray
```

In [2]:
```
# Define filepath
filepath = '/Users/jack/Documents/GitHub/geospatial-data-science/labs/

# Define list of Landsat bands
files = sorted(glob.glob(filepath + 'landsat/*.tif'))
print(files)
```

```
['/Users/jack/Documents/GitHub/geospatial-data-science/labs/lab4/land
sat/LC08_L2SP_047029_20200814_20210330_02_T1_SR_B1.tif', '/Users/jack
/Documents/GitHub/geospatial-data-science/labs/lab4/landsat/LC08_L2SP
_047029_20200814_20210330_02_T1_SR_B2.tif', '/Users/jack/Documents/Gi
tHub/geospatial-data-science/labs/lab4/landsat/LC08_L2SP_047029_20200
814_20210330_02_T1_SR_B3.tif', '/Users/jack/Documents/GitHub/geospati
al-data-science/labs/lab4/landsat/LC08_L2SP_047029_20200814_20210330_
02_T1_SR_B4.tif', '/Users/jack/Documents/GitHub/geospatial-data-scien
ce/labs/lab4/landsat/LC08_L2SP_047029_20200814_20210330_02_T1_SR_B5.t
if', '/Users/jack/Documents/GitHub/geospatial-data-science/labs/lab4/
landsat/LC08_L2SP_047029_20200814_20210330_02_T1_SR_B6.tif', '/Users/
jack/Documents/GitHub/geospatial-data-science/labs/lab4/landsat/LC08_
L2SP_047029_20200814_20210330_02_T1_SR_B7.tif', '/Users/jack/Document
s/GitHub/geospatial-data-science/labs/lab4/landsat/rgb.tif']
```

In [3]:
```
# Open a single band
src = rasterio.open(files[0])
band_1 = src.read(1)
```

In [4]:
```
# Find metadata (e.g. driver, data type, coordinate reference system,
print(src.profile)
```

```
{'driver': 'GTiff', 'dtype': 'uint16', 'nodata': 0.0, 'width': 1208,
'height': 1422, 'count': 1, 'crs': CRS.from_epsg(32610), 'transform':
Affine(30.0, 0.0, 391695.0,
       0.0, -30.0, 4880565.0), 'tiled': False, 'interleave': 'band'}
```

In [5]:
```
# Find coordinate reference system
src.crs # https://epsg.io/32610
```

Out[5]:
```
CRS.from_epsg(32610)
```

In [6]:
```
# Find format
src.driver
```

Out[6]:
```
'GTiff'
```

In [7]:
```
# Find pixel size
src.transform[0]
```

Out[7]:
```
30.0
```

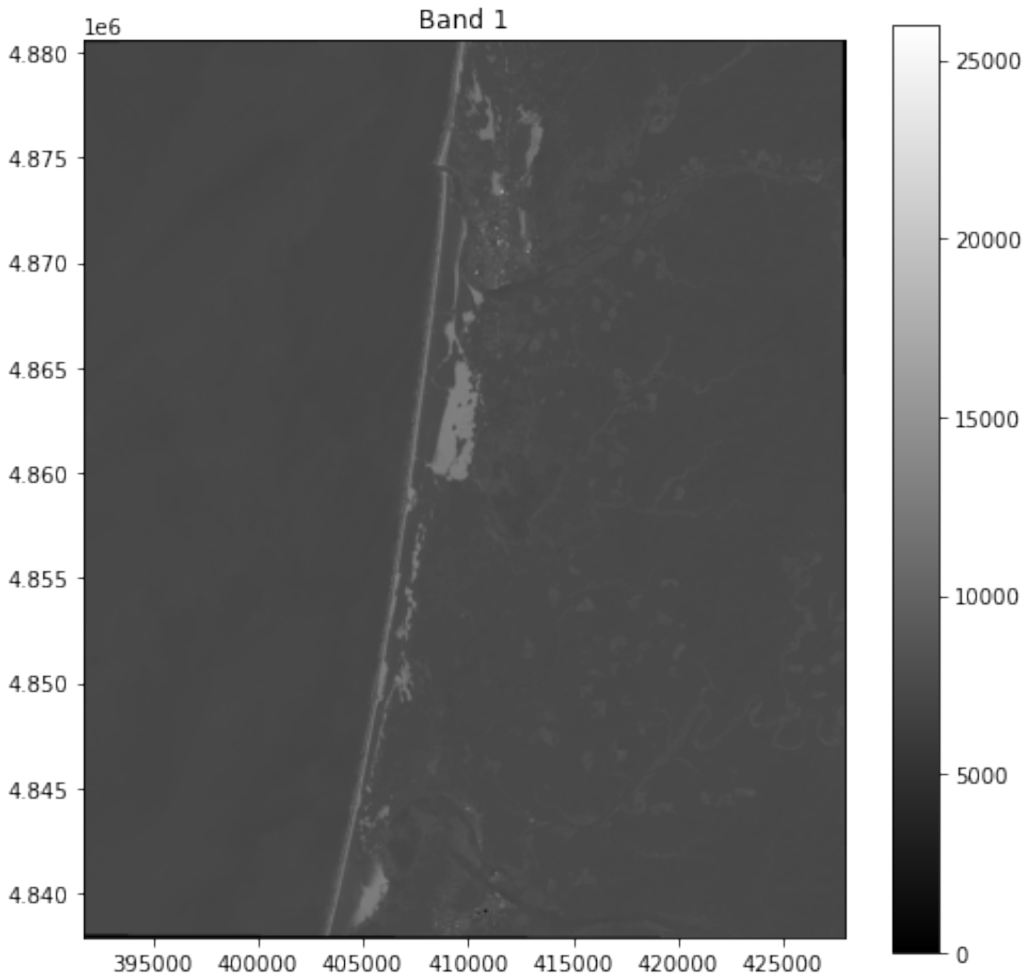
In [8]:
```
# Find bounds of dataset
src.bounds
```

Out[8]:
```
BoundingBox(left=391695.0, bottom=4837905.0, right=427935.0, top=4880
565.0)
```

In [9]:
```
# Get corners of dataset
full_extent = [src.bounds.left, src.bounds.right, src.bounds.bottom, s
print(full_extent)
```

```
[391695.0, 427935.0, 4837905.0, 4880565.0]
```

In [10]:
```
# Plot dataset
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(band_1, cmap='gray', extent=full_extent)
ax.set_title("Band 1")
fig.colorbar(im, orientation='vertical')
plt.show()
```



In [11]:
```
# Find number of columns and rows in array
band_1.shape
```

Out[11]: (1422, 1208)

In [12]:
```
# Find total number of pixels in array
band_1.size
```

Out[12]: 1717776

In [13]:
```
# Find maximum value in array
band_1.max()
```

Out[13]: 25983

In [14]:
```
# Find datatype
band_1.dtype
```

Out[14]: dtype('uint16')

In [15]:
```
# Find maximum possible value in array
2**16
```

Out[15]: 65536

In [16]:
```
# Find file size (in MB)
band_1.nbytes / 1000000
```

Out[16]: 3.435552

In [17]:
```
# Open all bands in a loop
list_bands = []
for file in files:
    # Read band
    src = rasterio.open(file)
    band = src.read(1)

    # Append to list
    list_bands.append(band)

# Convert from list of arrays to n-dimensional array
all_bands = np.dstack(list_bands)
```

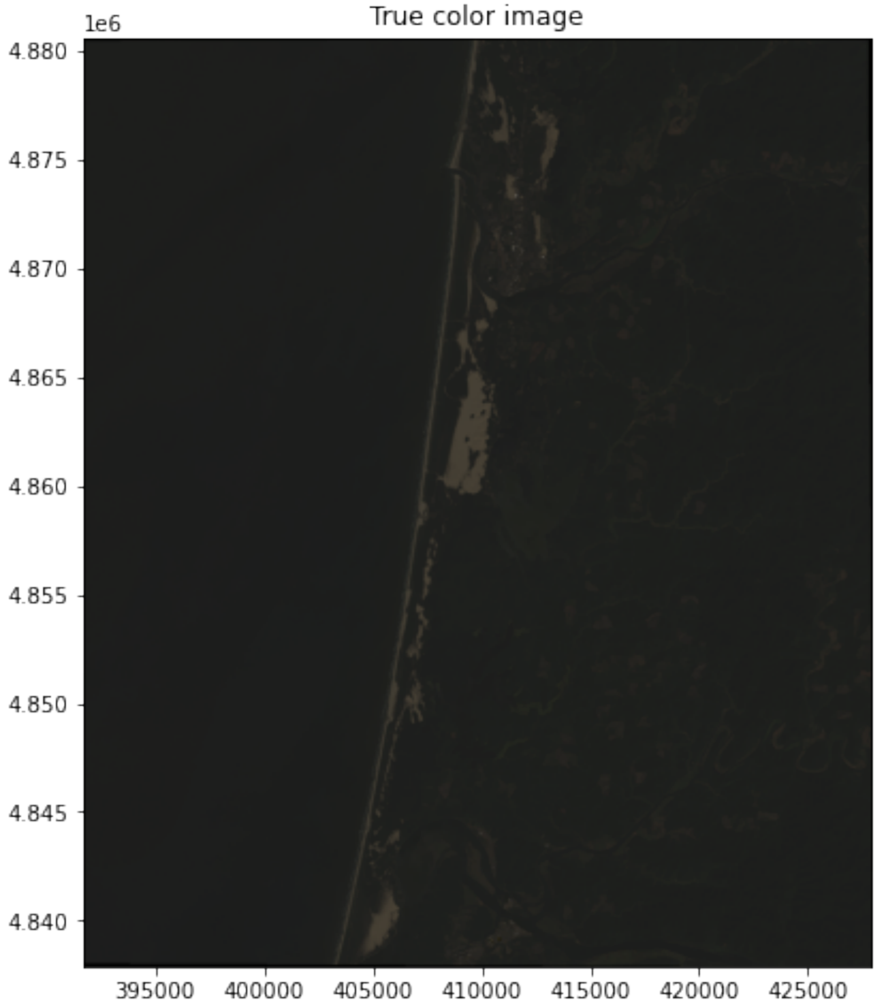
In [18]:
```
all_bands.shape
```

Out[18]: (1422, 1208, 8)

In [19]:
```
# Convert values to a range of 0-255
all_bands_image = np.uint8((all_bands / 65536) * 255)
```

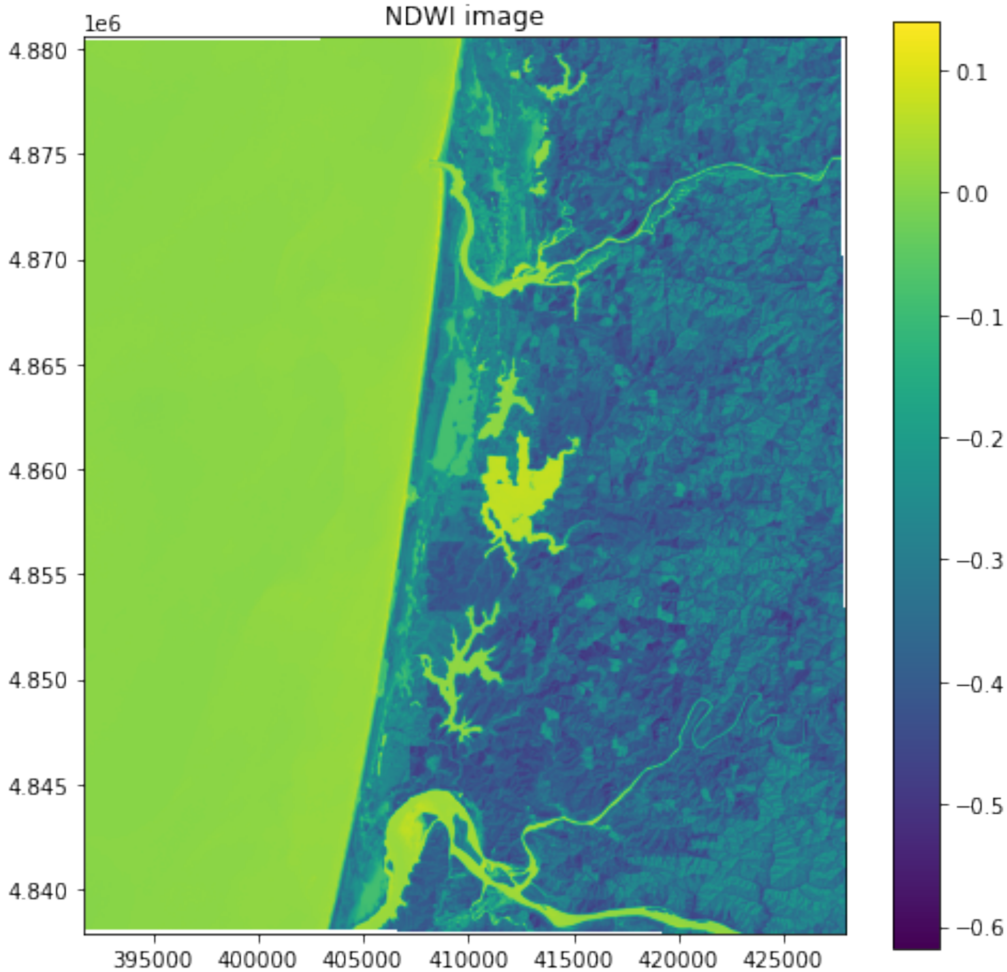
In [20]:
```
# Produce a new array by stacking the RGB bands
rgb = np.dstack((all_bands_image[:,:,3],all_bands_image[:,:,2],all_bar
```

In [21]:
```
# Plot as RGB image
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(rgb, extent=full_extent)
ax.set_title("True color image")
plt.show()
```



In [22]:
```
# Compute NDWI
np.seterr(divide='ignore', invalid='ignore')
ndwi = np.divide((all_bands[:,:,2].astype(float) - all_bands[:,:,4].as
                  (all_bands[:,:,2].astype(float) + all_bands[:,:,4].as
```

In [23]:
```
# Plot NDWI image
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(ndwi, extent=full_extent)
ax.set_title("NDWI image")
fig.colorbar(im, orientation='vertical')
plt.show()
```



In [24]:
```
# Write an array as a raster band to a new 8-bit file. For the new fil
# we start with the profile of the source
profile = src.profile

# And then change the band count to 3, set the dtype to uint8, and spe
profile.update(dtype=rasterio.uint8, count=3, compress='lzw')
```

In [25]:
```
with rasterio.open(filepath + 'landsat/rgb.tif', 'w', **profile) as ds
    # Write array
    dst.write(np.rollaxis(rgb, axis=2)) # Note that array needs to be
```

In [26]:
```
# Read data
xds = xarray.open_dataset(filepath + 'era/usa_t2m_tcc_2020.nc', decode
```

In [27]:
```
xds
```

Out[27]:

xarray.Dataset

▶ Dimensions: (**longitude**: 233, **latitude**: 99, **time**: 1464)

▼ Coordinates:

longitude	(longitude)	float32	-125.0 -124.8 ......
latitude	(latitude)	float32	49.24 48.99 48....
time	(time)	datetime64[ns]	2020-01-01 ... 2...

▼ Data variables:

t2m	(time, latitude, longitude)	float32	...
tcc	(time, latitude, longitude)	float32	...

▼ Attributes:

Conventions : CF-1.6

history : 2022-01-05 17:55:44 GMT by grib_to_netcdf-2.23.0: /opt/ecmwf/mars-client/bin/grib_to_netcdf -S param -o /cache/data7/adaptor.mars.internal-1641405337.2156463-13224-2-925a6819-f76e-4e12-a8ce-e9d715b345dd.nc /cache/tmp/925a6819-f76e-4e12-a8ce-e9d715b345dd-adaptor.mars.internal-1641405210.6756463-13224-3-tmp.grib

In [28]:
```
# Print the time period of the data
print('The data ranges from %s to %s' %(xds['t2m']['time'].values.min(
```

```
The data ranges from 2020-01-01T00:00:00.000000000 to 2020-12-31T18:0
0:00.000000000
```

In [29]:
```
xds_daily = xds.resample(time='1D').mean()
xds_daily
```

Out[29]:

xarray.Dataset

▶ Dimensions: (**time**: 366, **longitude**: 233, **latitude**: 99)

▼ Coordinates:

time	(time)	datetime64[ns]	2020-01-01 ... 2...
longitude	(longitude)	float32	-125.0 -124.8 ......
latitude	(latitude)	float32	49.24 48.99 48....

▼ Data variables:

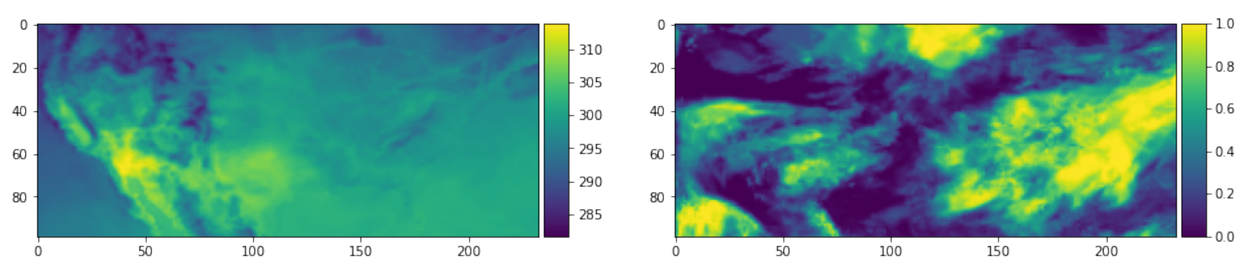
t2m	(time, latitude, longitude)	float32	280.6 281.4 28...
tcc	(time, latitude, longitude)	float32	0.9765 0.8814 ....

▶ Attributes: (0)

In [30]:
```
# Plot data
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(16,6))
im1 = ax1.imshow(xds_daily['t2m'][226,:,:])
divider = make_axes_locatable(ax1)
cax = divider.append_axes('right', size='5%', pad=0.05)
fig.colorbar(im1, cax=cax, orientation='vertical')

im2 = ax2.imshow(xds_daily['tcc'][226,:,:])
divider = make_axes_locatable(ax2)
cax = divider.append_axes('right', size='5%', pad=0.05)
fig.colorbar(im2, cax=cax, orientation='vertical')
```

Out[30]: <matplotlib.colorbar.Colorbar at 0x7f8823ca3ee0>



In [31]:
```
# Next, find the index of the grid point nearest a specific lat/lon.
florence_weather = xds_daily.sel(latitude=43.974659, longitude=-124.10
```

```
/Users/jack/opt/miniconda3/envs/lab4/lib/python3.8/site-packages/xarr
ay/core/indexes.py:234: FutureWarning: Passing method to Float64Index
.get_loc is deprecated and will raise in a future version. Use index.
get_indexer([item], method=...) instead.
  indexer = self.index.get_loc(
/Users/jack/opt/miniconda3/envs/lab4/lib/python3.8/site-packages/xarr
ay/core/indexes.py:234: FutureWarning: Passing method to Float64Index
.get_loc is deprecated and will raise in a future version. Use index.
get_indexer([item], method=...) instead.
  indexer = self.index.get_loc(
```

In [32]:
```
# Note: Aug 14 is DOY 226
print('Cloud cover in Florence on Aug 14, 2020 = %.2f %%' % (florence_
```

```
Cloud cover in Florence on Aug 14, 2020 = 0.02 %
```

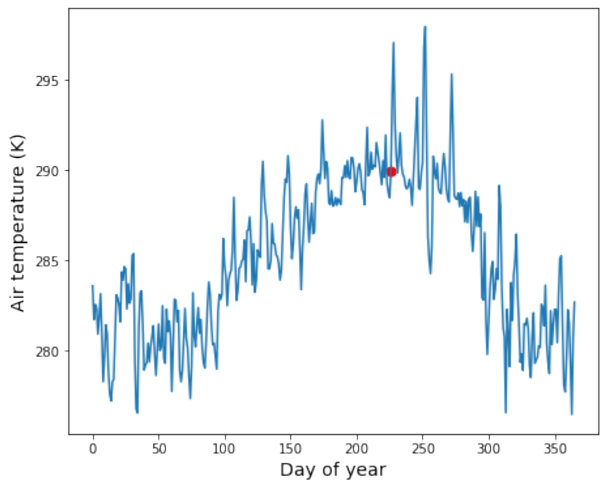
In [33]:
```
fahrenheit = (florence_weather['t2m'][226].values - 273.15) * 9/5 + 32
print('Air temperature in Florence on Aug 14, 2020 = %.2f F' % (fahren
```

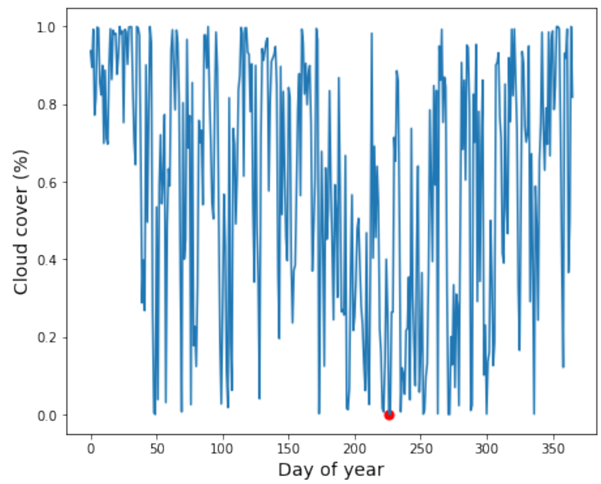
```
Air temperature in Florence on Aug 14, 2020 = 62.25 F
```

In [34]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(16,6))
ax1.plot(florence_weather['t2m'])
ax1.scatter(226, florence_weather['t2m'][226], s=50, color='r')
ax1.set_xlabel('Day of year', fontsize=14)
ax1.set_ylabel('Air temperature (K)', fontsize=14)

ax2.plot(florence_weather['tcc'])
ax2.scatter(226, florence_weather['tcc'][226], s=50, color='r')
ax2.set_xlabel('Day of year', fontsize=14)
ax2.set_ylabel('Cloud cover (%)', fontsize=14)
```

Out[34]:
```
Text(0, 0.5, 'Cloud cover (%)')
```





In [35]:
```
mean_temp = (florence_weather['t2m'].mean() - 273.15) * 9/5 + 32
print('Mean air temp. in Florence in 2020 = %.2f F' % (mean_temp))
```

```
Mean air temp. in Florence in 2020 = 53.55 F
```

In [36]:
```
#Find the index of the grid point nearest Eugene OR
eugene_weather = xds_daily.sel(latitude=44.0521, longitude=-123.0868,
```

```
/Users/jack/opt/miniconda3/envs/lab4/lib/python3.8/site-packages/xarr
ay/core/indexes.py:234: FutureWarning: Passing method to Float64Index
.get_loc is deprecated and will raise in a future version. Use index.
get_indexer([item], method=...) instead.
  indexer = self.index.get_loc(
/Users/jack/opt/miniconda3/envs/lab4/lib/python3.8/site-packages/xarr
ay/core/indexes.py:234: FutureWarning: Passing method to Float64Index
.get_loc is deprecated and will raise in a future version. Use index.
get_indexer([item], method=...) instead.
  indexer = self.index.get_loc(
```

In [37]:
```
mean_cloud = florence_weather['tcc'].mean()
print('Mean cloud cover in Florence in 2020 = %.2f %%' % (mean_cloud *
```

Mean cloud cover in Florence in 2020 = 58.76 %

In [38]:
```
days = np.sum(florence_weather['tcc'] < 0.2).values
print('There were %.0f days with less than 20%% cloud cover in 2020' %
```

There were 61 days with less than 20% cloud cover in 2020

Question 1 (10 points):

```
Now that we have gone through some examples in the lecture and lab we
are ready to apply some of these methods ourselves. Start by making a
new jupyter notebook called lab4_submission.ipynb and complete the
following tasks.
Find the following numbers in the climate reanalysis dataset:
a) the air temperature (in F) and cloud cover (in %) in Florence, OR
(in 2020) on January 31, 2020?
b) the air temperature (in F) and cloud cover (in %) in Eugene, OR
(in 2020) on February 15, 2020?
```

In [39]:
```
florence_jan_temp = (florence_weather['t2m'][30].values - 273.15) * 9/
print('January 31st air temp. in Florence in 2020 = %.2f F' % (florenc
```

January 31st air temp. in Florence in 2020 = 53.82 F

Question 1a: January 31st air temp. in Florence in 2020 is 53.82 F.

In [40]:
```
florence_jan_cloud = florence_weather['tcc'][30].values
print('Cloud cover in January 15 in Eugene in 2020 = %.2f %%' % (flore
```

Cloud cover in January 15 in Eugene in 2020 = 99.98 %

Question 1b: Cloud cover in January 15 in Eugene in 2020 is 99.98%.

In [41]:
```
eugene_feb_temp = (eugene_weather['t2m'][45].values - 273.15) * 9/5 +
print('February 15th air temp. in Eugene in 2020 = %.2f F' % (eugene_f
```

February 15th air temp. in Eugene in 2020 = 42.00 F

Question 1c: February 15th air temp. in Eugene in 2020 in 42.00 F.

In [42]:
```
eugene_feb_cloud = eugene_weather['tcc'][45].values
print('Cloud cover in February 15 in Eugene in 2020 = %.2f %%' % (euge
```

```
Cloud cover in February 15 in Eugene in 2020 = 99.99 %
```

Question 1d: Cloud cover in February 15 in Eugene in 2020 is 99.99%.

In [43]:
```
#Check date values
eugene_weather['tcc'][46]
```

Out[43]:

xarray.DataArray 'tcc'

```
array(0.9751423, dtype=float32)
```

▼ Coordinates:

time	()	datetime64[ns]	2020-02-16
longitude	()	float32	-123.0
latitude	()	float32	43.99

▶ Attributes: (0)

Question 2

```
Find the following grid cells in the climate reanalysis dataset and
provide the lat/lons and a rough location of where they are located.
a) Highest average air temperature (i.e. hottest place)
b) Lowest average air temperature (i.e. coldest place)
c) Highest average cloudiness (i.e. cloudiest place)
d) Lowest average cloudiest (i.e. least cloudy place)
e) Place with highest range in daily air temperature
f) Place with the absolute coldest temperature on a single day
```

In [44]:
```
mean = (xds_daily['t2m'].mean(dim='time')-273.15)*(9/5)+32
mean_max = mean.argmax()
print(mean_max)
```

```
<xarray.DataArray 't2m' ()>
array(18928)
```

In [45]:
```
high_idx = np.unravel_index(mean_max, xds_daily['t2m'].shape)
print(high_idx)
```

```
(0, 81, 55)
```

In [46]:
```
mean[81,55]
```

Out[46]:

xarray.DataArray 't2m'

```
array(79.06801, dtype=float32)
```

▼ Coordinates:

longitude	()	float32	-111.2
latitude	()	float32	28.99

► Attributes: (0)

Question 1a: The highest average air temperature is Plan de Ayala, Hermosillo, México

In [47]:
```
mean = (xds_daily['t2m'].mean(dim='time')-273.15)*(9/5)+32
mean_min = mean.argmin()
print(mean_min)
```

```
<xarray.DataArray 't2m' ()>
array(4954)
```

In [48]:
```
low_idx = np.unravel_index(mean_min, xds_daily['t2m'].shape)
print(low_idx)
```

```
(0, 21, 61)
```

In [49]:
```
mean[21,61]
```

Out[49]:

xarray.DataArray 't2m'

array(29.179205, dtype=float32)

▼ Coordinates:

longitude	()	float32	-109.8
latitude	()	float32	43.99

► Attributes: (0)

Question 1b: The lowest average air temperature is Shoshone National Forest, Cody, Wymoing.

In [50]:
```
mean = xds_daily['tcc'].mean(dim='time')
mean_max = mean.argmax()
print(mean_max)
```

```
<xarray.DataArray 'tcc' ()>
array(0)
```

In [51]:
```
high_idx = np.unravel_index(mean_max, xds_daily['tcc'].shape)
print(high_idx)
```

```
(0, 0, 0)
```

In [52]:
```
(mean[0,0])
```

Out[52]:

xarray.DataArray 'tcc'

array(0.77146894, dtype=float32)

▼ Coordinates:

longitude	()	float32	-125.0
latitude	()	float32	49.24

► Attributes: (0)

Question 1c: The highest average cloudiness is in Two Rivers Arm, Alberni-Clayoquot Regional District BC, Canada.

In [53]:
```
mean = xds_daily['tcc'].mean(dim='time')
mean_min = mean.argmin()
print(mean_max)
```

```
<xarray.DataArray 'tcc' ()>
array(0)
```

In [54]:
```
high_idx = np.unravel_index(mean_min, xds_daily['tcc'].shape)
print(high_idx)
```

```
(0, 71, 41)
```

In [55]:
```
(mean[71,41])
```

Out[55]:

xarray.DataArray 'tcc'

```
array(0.16893195, dtype=float32)
```

▼ Coordinates:

longitude	()	float32	-114.8
latitude	()	float32	31.49

▶ Attributes: (0)

Question 1d: The lowest average cloudiness is Baja California, Mexico.

In [56]:
```
daily_max = np.max(xds_daily['t2m'],axis=0)
daily_max
daily_min = np.min(xds_daily['t2m'],axis=0)
daily_min
temprange = (daily_max - daily_min)
temprange
high_range = temprange.argmax()
high_range
high_range_idx = np.unravel_index(high_range,xds_daily['t2m'].shape)
print(high_range_idx)
```

```
(0, 1, 210)
```

In [57]:
```
(mean[1,210])
```

Out[57]:

xarray.DataArray 'tcc'

```
array(0.6921238, dtype=float32)
```

▼ Coordinates:

longitude	()	float32	-72.5
latitude	()	float32	48.99

▶ Attributes: (0)

Question 1e: The place with highest range in daily air temperature is Girardville QC, Canada.

In [58]:
```
daily_max = np.max(xds_daily['t2m'],axis =0)
daily_min = np.min(xds_daily['t2m'],axis =0)
temp_range = daily_max - daily_min
max_range = temp_range.argmax()
max_range_index = np.unravel_index(max_range, np.max(xds_daily['t2m'],
max_range_index
max_range_loc = np.max(xds_daily['t2m'], axis=0)[max_range_index]
max_range_loc

min_value = daily_min.argmin()
min_value_index = np.unravel_index(min_value, np.min(xds_daily['t2m'],
min_value_loc = np.max(xds_daily['t2m'], axis = 0)
min_value_loc
```

Out[58]:

xarray.DataArray 't2m' (**latitude**: 99, **longitude**: 233)

```
array([[294.49335, 294.26083, 294.98743, ..., 293.40433, 293.8870
5,
        294.4807 ],
       [294.85352, 294.37445, 293.87637, ..., 294.7832 , 294.5513
3,
        294.73587],
       [292.5385 , 293.26373, 293.71115, ..., 300.5082 , 300.6311
3,
        299.97418],
       ...,
       [296.69775, 296.6911 , 296.75406, ..., 302.0663 , 302.0553
,
        302.08466],
       [296.8783 , 296.93097, 296.95294, ..., 302.03665, 301.9913
6,
        302.025  ],
       [297.1285 , 297.12918, 297.1708 , ..., 301.992  , 301.9577
,
        301.9667 ]], dtype=float32)
```

▼ Coordinates:

longitude	(longitude)	float32	-125.0 -124.8 ... -67.25 -67.0
latitude	(latitude)	float32	49.24 48.99 48.74 ... 24.99 24.74

► Attributes: (0)

In [ ]:

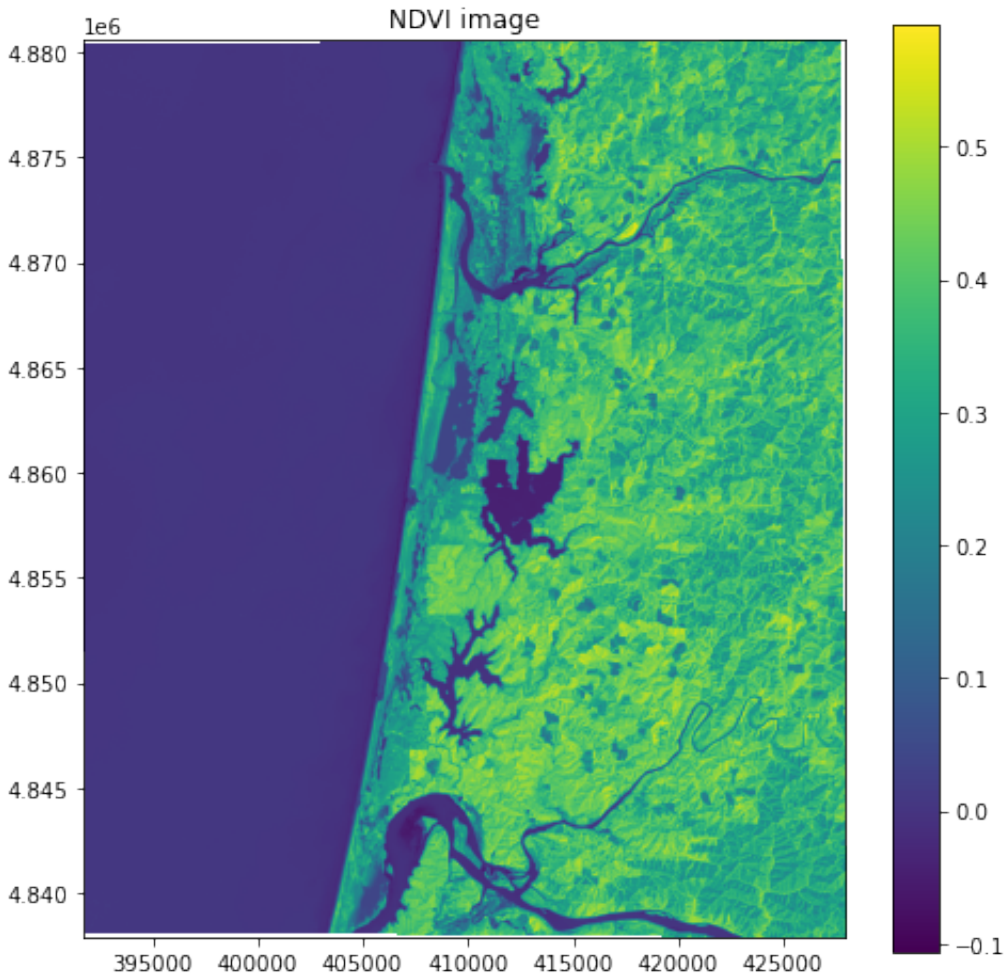
Question 1f: The place with the absolute coldest temperature on a single day is Whitlash, Montana USA.

Question 3

a) an NDVI image (i.e. (Band 5 - Band 4) / (Band 5 + Band 4))

In [59]:
```
# Compute NDVI
np.seterr(divide='ignore', invalid='ignore')
ndwi = np.divide((all_bands[:,:,4].astype(float) - all_bands[:,:,3].as
                  (all_bands[:,:,4].astype(float) + all_bands[:,:,3].as
```

In [60]:
```
# Plot NDVI image
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(ndwi, extent=full_extent)
ax.set_title("NDVI image")
fig.colorbar(im, orientation='vertical')
plt.show()
```



b) a color infrared composite (i.e. bands 5, 4, 3)

In [61]:
```
# Convert values to a range of 0-255
all_bands_image = np.uint8((all_bands / 65536) * 255)
```

In [62]:
```
# Produce a new array by stacking the RGB bands
rgb = np.dstack((all_bands_image[:,:,4],all_bands_image[:,:,3],all_bar
```

In [63]:
```
# Plot as RGB image
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(rgb, extent=full_extent)
ax.set_title("Color Infrared Composite")
plt.show()
```