Scientific Computing: Coursework Part 1

This report aims to provide assessment of environmental variables, specifically discharge, temperature, and snow cover at the Del Norte monitoring station of Colorado, USA. A further purpose of this report is the preparation of data collected at the Del Norte site for future hydrological modelling utilising datasets of the aforementioned variables.

Data will be sourced from http://climate.colostate.edu (temperature) and http://waterdata.usgs.gov (discharge). An assessment of the HUC catchment 13010001 Rio Grande headwaters (Colorado, USA) will be undertaken to understand snow cover in the region. Data in this study will be sourced from: https://nsidc.org/data/MOD10A1

Some basic geographical data on the station is given below:

Station Name: DEL NORTE 2E Latitude: -106.30833 Longitude: 37.69083 Elevation: 7845 ft.

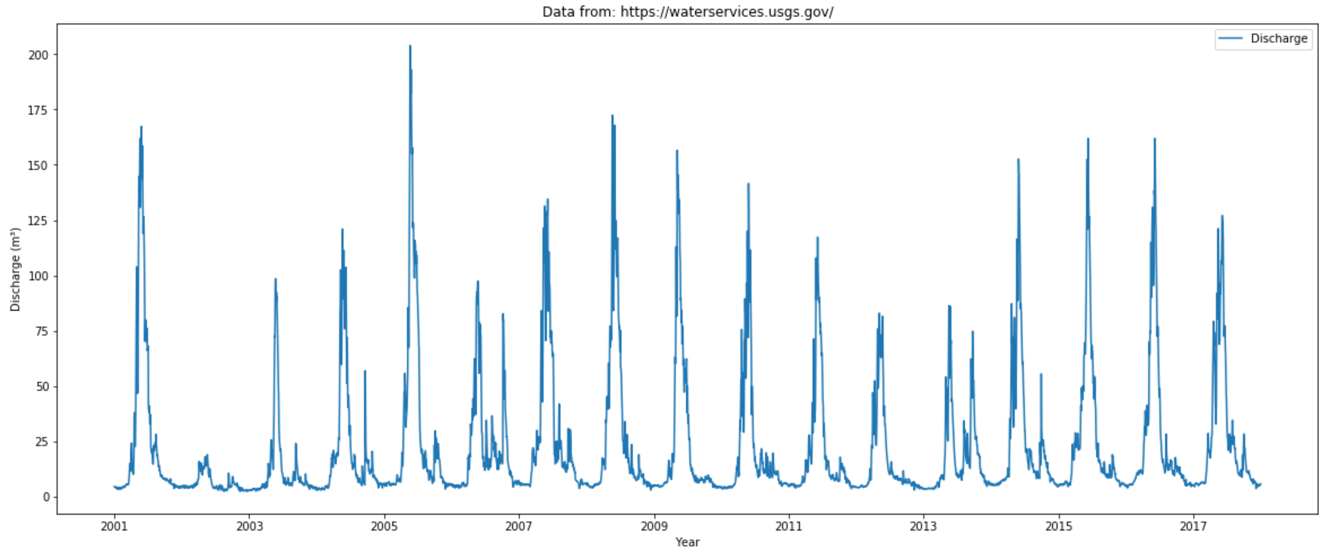
Discharge at Del Norte Monitoring Station

In [192]:

```
# Necessary module imports.

import requests
import datetime
import numpy as np
from io import StringIO
import matplotlib.pyplot as plt
%matplotlib inline

# Given url from practical instructions.

url = "https://waterservices.usgs.gov/nwis/dv/?sites=08220000&format=rdb&startDT=2001-01-
01&parameterCd=00060"

discharge_dataset = requests.get(url).text

# Having opened the text file from the given url, the first 30 lines are background information or
headings.
# Grabbing the discharge data in column 3 into an array.
discharge = np.loadtxt(StringIO(discharge_dataset), skiprows=30, usecols=[3], delimiter="\t", unpac
k=True)
# Grabbing the string formatted dates in column 2 into an array.
dates = np.loadtxt(StringIO(discharge_dataset), dtype=str, skiprows=30, usecols=[2], delimiter="\t"
, unpack=True)

# Convert dates into a datetime friendly format:
dates = [date.replace("-", "/") for date in dates]

# Converting the string formatted dates array for compatiblity with matplotlib.
# Information on using strptime in this form was taken from:
# https://stackoverflow.com/questions/9627686/plotting-dates-on-the-x-axis-with-pythons-matplotlib
x_axis = [datetime.datetime.strptime(date,'%Y/%m/%d').date() for date in dates]

# Converting discharge into metric units from cubic feet given below ratio.
conversion_to_cubicm = 0.0283168

discharge = discharge * conversion_to_cubicm

# Plotting a figure for all years of discharge measured.
plt.figure(figsize=(20,8))
plt.plot(x_axis,discharge,label='Discharge')

plt.title(f"Data from: {url[0:31]}")
plt.xlabel('Year')
plt.ylabel('Discharge (m³)')
plt.legend(loc='best')
```

Out[192]:

```
<matplotlib.legend.Legend at 0x7efc3dcbc320>
```



In [193]:

```
import pandas as pd

# Creating a list of strings in the format 2001/01/01 which can be used to find index values.
years = []
for i in range(2002,2018):
    years.append(str(i)+"/01/01")

# Creating a loop to run through the dates array and return the index of the start of each year.

indices = []
for year in years:
    indices.append(dates.index(year))


# Converting the above placeholder list into an array.
indices = np.asarray(indices)

# Splitting the discharge array into yearly sub-arrays based on the indexed years formed above.
discharge_year = np.split(discharge, indices, axis=0)

# Creating placeholder lists for summary statistics.

mean = []
mini = []
argmin = []
maxi = []
argmax = []
stdev = []

# Looping through sub arrays to perform summary statistics.

for i in range(0, 17):
    mean.append(discharge_year[i].mean())
    mini.append(discharge_year[i].min())
    argmin.append(discharge_year[i].argmin())
    maxi.append(discharge_year[i].max())
    argmax.append(discharge_year[i].argmax())
    stdev.append(discharge_year[i].std())

# Display options for pandas dataframe table, specifically to reduce floats to 2dp where appropria
te.

pd.options.display.float_format = '{:,.2f}'.format


print("Summary statistics for discharge (m³) at Del Norte monitoring station, Colorado 2001-17:" )

# Creating a table using pandas to display summary statistics

df = pd.DataFrame({ "Year": range(2001,2018),
                    "Mean": mean,
                   "StDev": stdev,
```

```
                    "Minimum": mini,
                    "Minimum Time (doy)": argmin,
                    "Maximum": maxi,
                    "Maximum Time (doy)": argmax
                    })

df
```

Summary statistics for discharge (m³) at Del Norte monitoring station, Colorado 2001-17:

Out[193]:

	Year	Mean	StDev	Minimum	Minimum Time (doy)	Maximum	Maximum Time (doy)
0	2001	28.37	40.65	3.40	16	167.35	147
1	2002	6.03	3.77	2.49	227	19.00	139
2	2003	12.49	18.63	2.55	37	98.54	143
3	2004	20.59	26.45	2.83	4	120.91	141
4	2005	31.05	43.17	3.45	332	203.88	141
5	2006	22.30	22.33	3.96	20	97.41	142
6	2007	27.78	32.07	3.68	327	134.50	156
7	2008	27.70	36.03	2.97	349	172.45	141
8	2009	23.20	33.32	3.96	343	156.59	127
9	2010	21.10	27.08	3.40	8	141.58	148
10	2011	19.66	25.06	3.68	339	117.23	157
11	2012	15.87	19.09	2.97	316	82.97	127
12	2013	17.98	17.90	3.40	4	86.37	137
13	2014	24.98	30.17	4.39	361	152.63	149
14	2015	26.01	31.63	4.53	53	161.97	161
15	2016	26.04	33.68	3.96	9	161.97	157
16	2017	27.00	30.19	3.68	339	127.14	156

Having looked at the data for all years plotted and summarised statistically above, the years 2016 and 2017 will be selected as they display a fairly typical seasonal discharge cycle but there is some variance between the years which will make for an interesting comparison between the two. Sequential years have been chosen so that changes to the system are less likely to be effected by wider changes in climate which might occur over a span of ten or more years.

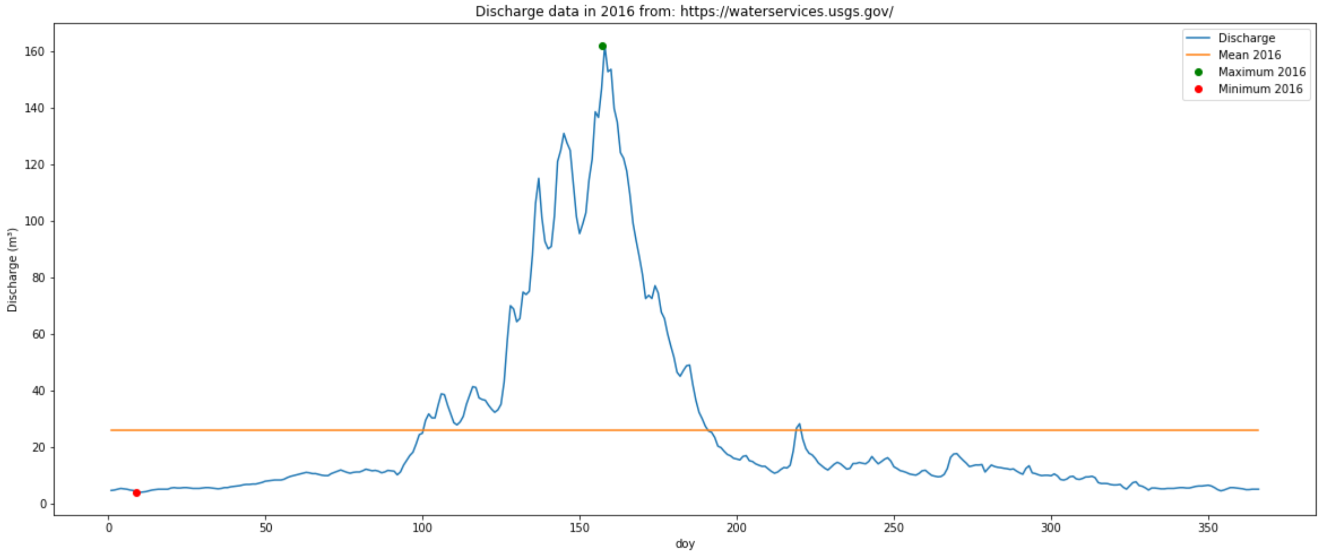
Discharge at Del Norte Monitoring Station (2016/17)

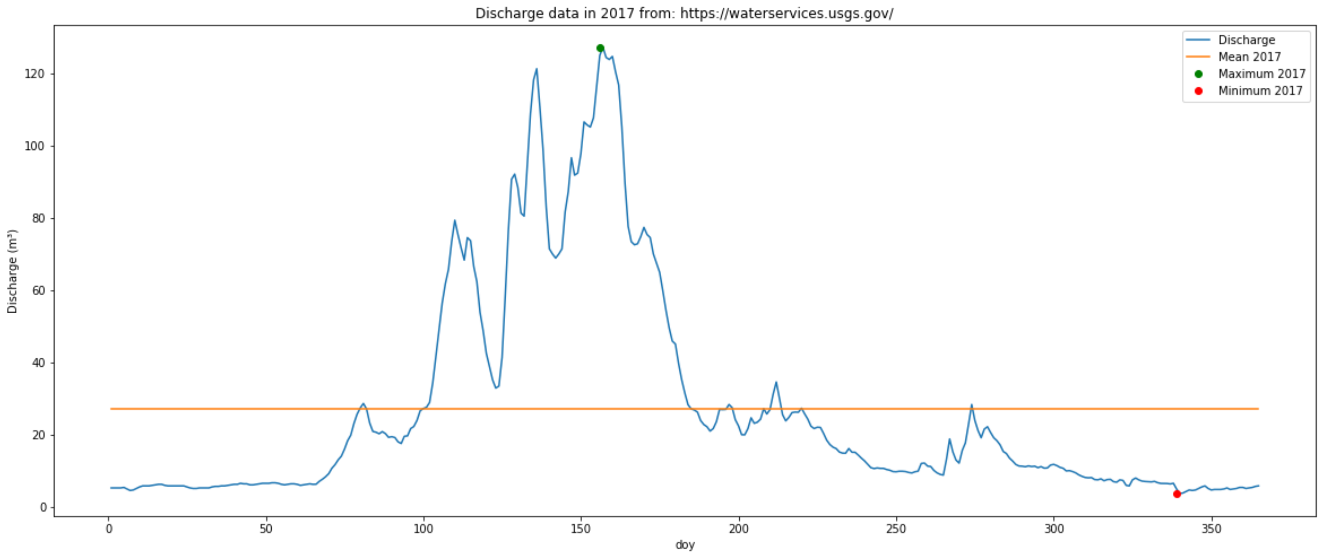
In [194]:

```
# Plotting individual graphs of discharge for each of the selected years.

print("Individual plots of seasonal discharge for the years 2016/17:")

for i in range(15,17):

    plt.figure(figsize=(20,8))
    # Creating a dynamic x-axis/doy based on the length of the dataset.
    # Carried backwards from later plots where comparing arrays of two different lengths (leap-yea
rs) required...
    # A dynamic axis.
    x_axis = np.arange(1,(len(discharge_year[i])+1))
    plt.plot(x_axis, discharge_year[i], label="Discharge")
    # Correctly creating an axis for a mean (fixed value) line taken from the notes of Exercise 2.
5.2
    plt.plot([x_axis[0], x_axis[-1]], [mean[i], mean[i]], label="Mean " + str(2001+i))
    # Plotting min and max values by doy, although maximum may be obvious.
    plt.plot(argmax[i], maxi[i], "go", label="Maximum "+ str(2001+i), color = "g")
```

```
    plt.plot(argmin[i], mini[i], "go", label="Minimum " + str(2001+i), color = "r")

    plt.title("Discharge data in " + str(2001+i) + f" from: {url[0:31]}")
    plt.xlabel('doy')
    plt.ylabel('Discharge (m³)')
    plt.legend(loc='best')
```

Individual plots of seasonal discharge for the years 2016/17:

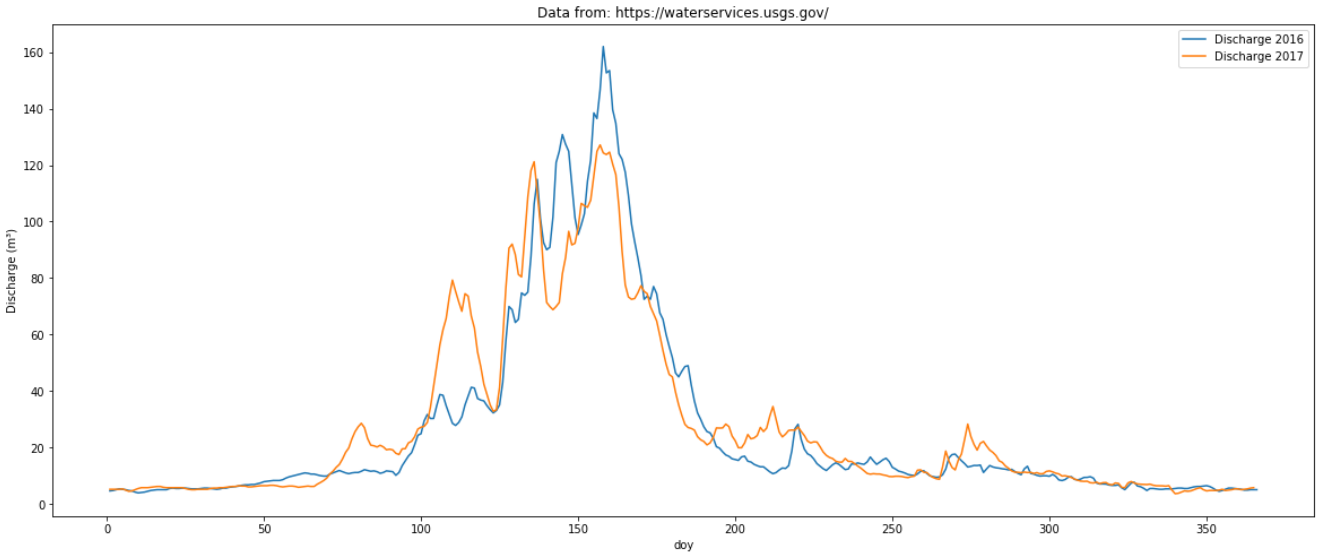




In [195]:

```
# Plotting a grouped graph of discharge for each of the selected years.

print("Shared plot of seasonal discharge for the years 2016/17:")

plt.figure(figsize=(20,8))

for i in range(15,17):

    # Creating an x-axis in this way to compensate for the extra day in 2016 whilst being able to.
..
    # plot the 2 years together
    x_axis = np.arange(1,(len(discharge_year[i])+1))
    plt.plot(x_axis, discharge_year[i], label="Discharge " + str(2001+i))

    plt.title(f"Data from: {url[0:31]}")
    plt.xlabel('doy')
    plt.ylabel('Discharge (m³)')
    plt.legend(loc='best')
```

Shared plot of seasonal discharge for the years 2016/17:



Conclusions:

Whilst the years 2016 and 2017 saw fairly similar patterns of discharge behavior at the monitoring site, there are some notable differences. Mean discharge was lower in 2016 at 26.04 (stdev 3.96) in comparison to 27.00 (stdev 3.68) in 2017 although the peak in early summer was more pronounced in 2016 at 167.97 compared to 127.14.

As shown in the graph above the 2017 cycle had a few more signifcant instances of early pulse events as well as smaller events after doy 200 which may be responsible for the mean discharge over the course of the year being increased.

In [223]:

```
# Saving the datasets.

# header = ["discharge"]

# use zip to load into a dictionary

discharge_2016 = discharge_year[15]
discharge_2017 = discharge_year[16]

filename16 = "discharge_DN_2016.npz"
filename17 = "discharge_DN_2017.npz"

np.savez_compressed(filename16,discharge_2016)
np.savez_compressed(filename17,discharge_2017)
```

Temperature at Del Norte Monitoring Station

Data for maximum temperature, minimum temperature, precipitation, and snowfall were downloaded from http://climate.colostate.edu/data_access.html for the site Del Norte 2E to cover the period ranging 2016/01/01 - 2017/12/31.

Whilst the precipitation and snowfall data may not be immediately useful there may be some basic comparative and modelling applications when looking at the relationships between discharge, snow cover, and temperature.

After downloading and following some basic formatting in Excel the data for each year was saved down in .csv format in order to be compatible for loading as an array.

In [197]:

```
# Assigning the given filenames to variables.

filename_2016 = "Del_Norte2E_2016.csv"
filename_2017 = "Del_Norte2E_2017.csv"

# Loading in the data from the .csv, and converting the date column into the datetime format.
```

```
DN2E2016 = np.loadtxt("data/"+filename_2016, dtype=object, skiprows=1, delimiter=",", unpack=True)

DN2E2016[0] = [date.replace("-", "/") for date in DN2E2016[0]]
DN2E2016[0] = [datetime.datetime.strptime(date,'%Y/%m/%d').date() for date in DN2E2016[0]]

DN2E2017 = np.loadtxt("data/"+filename_2017, dtype=object, skiprows=1, delimiter=",", unpack=True)

DN2E2017[0] = [date.replace("-", "/") for date in DN2E2017[0]]
DN2E2017[0] = [datetime.datetime.strptime(date,'%Y/%m/%d').date() for date in DN2E2017[0]]

# Using genfromtxt to replace placeholder values in dataset where data was missing or incomplete,
with NaN.

for i in range (1, 5):
    DN2E2016[i] = np.genfromtxt(DN2E2016[i])

for i in range (1, 5):
    DN2E2017[i] = np.genfromtxt(DN2E2017[i])

# Splitting array into individual parts for statistical analysis.

dates2016 = DN2E2016[0]

maxtemp2016 = DN2E2016[1]
# Converting to Celcius from fahrenheit.
maxtemp2016 = (maxtemp2016 -32)*5/9

mintemp2016 = DN2E2016[2]
# Converting to Celcius from fahrenheit.
mintemp2016 = (mintemp2016 -32)*5/9

pcpn2016 = DN2E2016[3]
snow2016 = DN2E2016[4]

# Duplicating process for 2017

dates2017 = DN2E2017[0]

maxtemp2017 = DN2E2017[1]
# Converting to Celcius from fahrenheit.
maxtemp2017 = (maxtemp2017 -32)*5/9

mintemp2017 = DN2E2017[2]
# Converting to Celcius from fahrenheit.
mintemp2017 = (mintemp2017 -32)*5/9

pcpn2017 = DN2E2017[3]
snow2017 = DN2E2017[4]

# Creating a list of dataset names of the above variables to use in the following for loop.
datasets = [maxtemp2016, mintemp2016, pcpn2016, snow2016, maxtemp2017, mintemp2017, pcpn2017, snow2
017]

mean = []
mini = []
argmin = []
maxi = []
argmax = []
stdev = []

# As above,creating a list of indexed statistics for each year and each of the downloaded variable
s.
# Use of np.nan statistics where necessary to avoid using NaN values in calculations.

for dataset in datasets:
    mean.append(np.nanmean(dataset, axis=0, dtype=float))
    mini.append(np.nanmin(dataset, axis=0))
    argmin.append(dataset.argmin())
    maxi.append(np.nanmax(dataset, axis=0))
    argmax.append(dataset.argmax())
    stdev.append(np.nanstd(dataset, axis=0, dtype=float))
```

Summary Statistics - Temperature:

In [198]:

```
print("Summary statistics for Maximum temperature (°C) at Del Norte 2E monitoring station, Colorad
o 2016-17:" )

# Creating a table using pandas to display summary statistics

df2 = pd.DataFrame({ "Year": range(2016,2018),
                    "Mean Max Temp" : mean[0:5:4],
                    "StDev": stdev[0:5:4],
                    "Lowest Max Temp": mini[0:5:4],
                    "Low Time (doy)": argmin[0:5:4],
                    "Highest Max Temp": maxi[0:5:4],
                    "High Time (doy)": argmax[0:5:4],
                   })

df2
```

```
Summary statistics for Maximum temperature (°C) at Del Norte 2E monitoring station, Colorado 2016-
17:
```

Out[198]:

	Year	Mean Max Temp	StDev	Lowest Max Temp	Low Time (doy)	Highest Max Temp	High Time (doy)
0	2016	15.22	10.12	-11.67	9	31.67	171
1	2017	15.76	9.25	-10.56	7	31.11	174

In [199]:

```
print("Summary statistics for Minimum temperature (°C) at Del Norte 2E monitoring station, Colorad
o 2016-17:" )

df3 = pd.DataFrame({ "Year": range(2016,2018),
                    "Mean Min Temp" : mean[1:6:4],
                    "StDev": stdev[1:6:4],
                    "Lowest Min Temp": mini[1:6:4],
                    "Low Time (doy)": argmin[1:6:4],
                    "Highest Min Temp": maxi[1:6:4],
                    "High Time (doy)": argmax[1:6:4]
                   })

df3
```

```
Summary statistics for Minimum temperature (°C) at Del Norte 2E monitoring station, Colorado 2016-
17:
```

Out[199]:

	Year	Mean Min Temp	StDev	Lowest Min Temp	Low Time (doy)	Highest Min Temp	High Time (doy)
0	2016	-1.55	8.24	-20.56	11	12.78	215
1	2017	-0.68	7.73	-19.44	6	13.89	205

In [200]:

```
print("Summary statistics for Precipitation (m) at Del Norte 2E monitoring station, Colorado 2016-
17:" )

df4 = pd.DataFrame({ "Year": range(2016,2018),
                    "Mean Pcpn" : mean[2:7:4],
                    "StDev": stdev[2:7:4],
                    "Lowest Pcpn": mini[2:7:4],
                    "Highest Pcpn": maxi[2:7:4],
                    "High (doy)": argmax[2:7:4]
                   })

df4
```

```
Summary statistics for Precipitation (m) at Del Norte 2E monitoring station, Colorado 2016-17:
```

Out[200]:

	Year	Mean Pcpn	StDev	Lowest Pcpn	Highest Pcpn	High (doy)
0	2016	0.03	0.09	0.00	0.72	106
1	2017	0.03	0.10	0.00	0.88	0

In [201]:

```python
print("Summary statistics for Snowfall (m) at Del Norte 2E monitoring station, Colorado 2016-17:"
)

df4 = pd.DataFrame({ "Year": range(2016,2018),
                    "Mean Snowfall" : mean[3:8:4],
                    "StDev": stdev[3:8:4],
                    "Lowest Snowfall": mini[3:8:4],
                    "Highest Snowfall": maxi[3:8:4],
                    "High (doy)": argmax[3:8:4]
                   })

df4
```

```
Summary statistics for Snowfall (m) at Del Norte 2E monitoring station, Colorado 2016-17:
```

Out[201]:

	Year	Mean Snowfall	StDev	Lowest Snowfall	Highest Snowfall	High (doy)
0	2016	0.13	0.64	0.00	6.80	106
1	2017	0.12	0.72	0.00	7.80	0

Plotting Temperature:

In [202]:

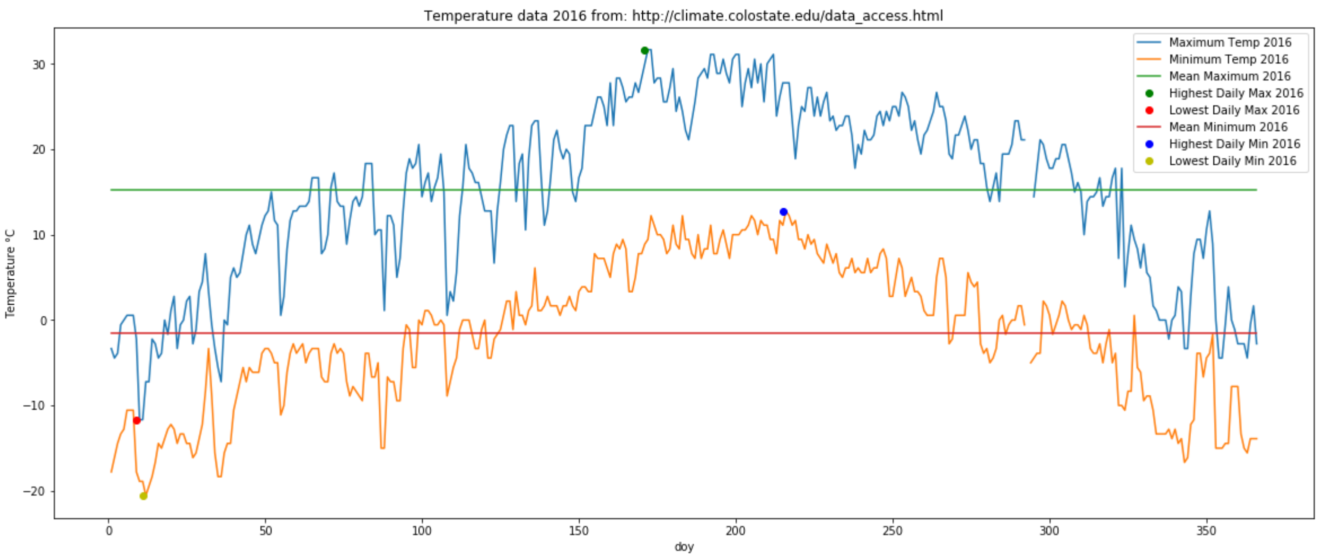
```python
url = "http://climate.colostate.edu/data_access.html"

print("Please note: Gaps in data are due to missing measurements replaced by NaN values.")

# Creating a plot for temperature 2016.

plt.figure(figsize=(20,8))
# Creating a dynamic x-axis/doy based on the length of the dataset.
x_axis = np.arange(1,(len(dates2016)+1))
plt.plot(x_axis, maxtemp2016, label="Maximum Temp 2016")
plt.plot(x_axis, mintemp2016, label="Minimum Temp 2016")
# Correctly creating an axis for a mean (fixed value) line taken from the notes of Exercise 2.5.2
plt.plot([x_axis[0], x_axis[-1]], [mean[0], mean[0]], label= "Mean Maximum 2016")
# Plotting min and max values by doy, although it may be obvious.
plt.plot(argmax[0], maxi[0], "go", label="Highest Daily Max 2016", color = "g")
plt.plot(argmin[0], mini[0], "go", label="Lowest Daily Max 2016", color = "r")

plt.plot([x_axis[0], x_axis[-1]], [mean[1], mean[1]], label= "Mean Minimum 2016")
# Plotting min and max values by doy, although it may be obvious.
plt.plot(argmax[1], maxi[1], "go", label="Highest Daily Min 2016", color = "b")
plt.plot(argmin[1], mini[1], "go", label="Lowest Daily Min 2016", color = "y")

plt.title("Temperature data 2016" + f" from: {url}")
plt.xlabel('doy')
plt.ylabel('Temperature °C')
plt.legend(loc='best')

# Creating a plot for temperature 2017.

plt.figure(figsize=(20,8))
# Creating a dynamic x-axis/doy based on the length of the dataset.
```

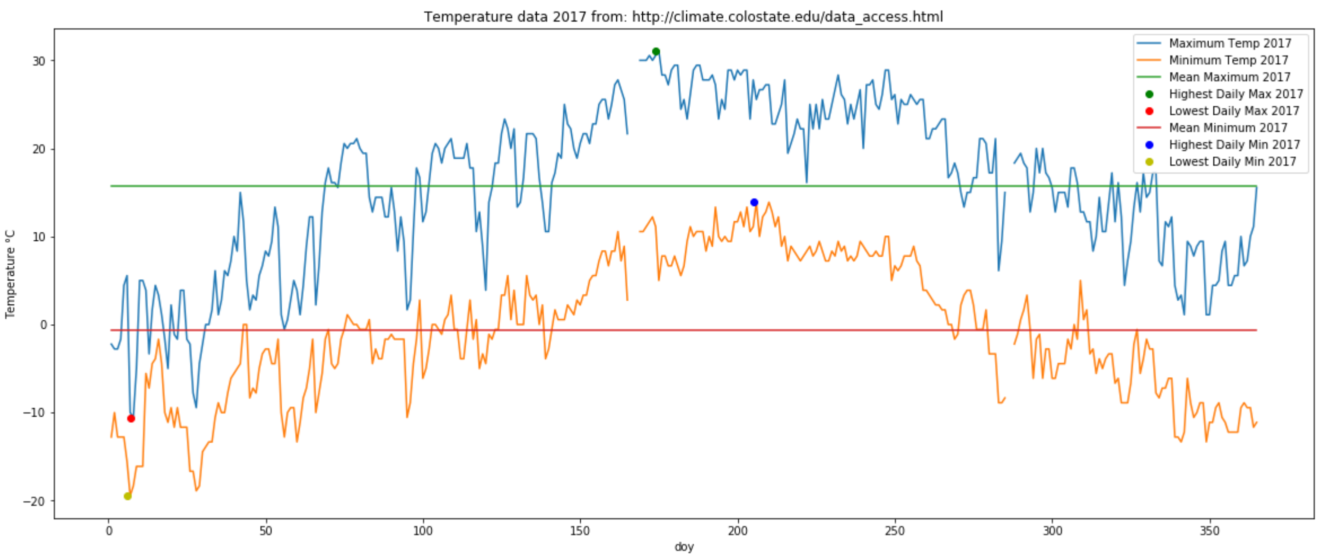
```
x_axis = np.arange(1,(len(dates2017)+1))
plt.plot(x_axis, maxtemp2017, label="Maximum Temp 2017")
plt.plot(x_axis, mintemp2017, label="Minimum Temp 2017")
# Correctly creating an axis for a mean (fixed value) line taken from the notes of Exercise 2.5.2
plt.plot([x_axis[0], x_axis[-1]], [mean[4], mean[4]], label= "Mean Maximum 2017")
# Plotting min and max values by doy, although it may be obvious.
plt.plot(argmax[4], maxi[4], "go", label="Highest Daily Max 2017", color = "g")
plt.plot(argmin[4], mini[4], "go", label="Lowest Daily Max 2017", color = "r")

plt.plot([x_axis[0], x_axis[-1]], [mean[5], mean[5]], label= "Mean Minimum 2017")
# Plotting min and max values by doy, although it may be obvious.
plt.plot(argmax[5], maxi[5], "go", label="Highest Daily Min 2017", color = "b")
plt.plot(argmin[5], mini[5], "go", label="Lowest Daily Min 2017", color = "y")

plt.title("Temperature data 2017" + f" from: {url}")
plt.xlabel('doy')
plt.ylabel('Temperature °C')
plt.legend(loc='best')
```

```
Please note: Gaps in data are due to missing measurements replaced by NaN values.
```

Out[202]:

```
<matplotlib.legend.Legend at 0x7efc38a75ef0>
```





Conclusions:

It is clear from the above graphical output and summary tables that the years 2016 and 2017 experienced fairly similar seasonal cycles at the Del Norte 2E station. Notably 2016 was on average a degree colder than 2017 - although this could be explained by missing data in the summer period of the latter - as well as experiencing a marked lower lowest minimum temperature and lowest maximum temperature. The range of temperatures experienced at the station is also quite remarkable, presumably due to the mountainous climate and elevation there is a real annual flux in the maximum daily temperatures recorded of around 30 degrees, and minimum temperatures of 20 degrees below freezing in the winter.

In [230]:

```
# Saving the datasets.

# header = ["doy", "maximum temp", "minimum temp", "precipitation", "snowfall"]

# use zip to load into a dictionary
DN2E_TPS_2016 = DN2E2016
DN2E_TPS_2017 = DN2E2017

filename16 = "DN2E_TPS_2016.npz"
filename17 = "DN2E_TPS_2017.npz"

np.savez_compressed(filename16,DN2E_TPS_2016)
np.savez_compressed(filename17,DN2E_TPS_2017)
```

Snow Cover For HUC Catchment 13010001

In [300]:

```
import datetime

# Defining a tile and base url for snow cover data download.

tile  = 'h09v05'
base_url = 'https://n5eil01u.ecs.nsidc.org/MOST/MOD10A1.006/'

#
start = datetime.datetime(2016,1,1)
urls = [base_url  + (start + datetime.timedelta(i)).strftime('%Y.%m.%d') + '/' for i in range(0,731
)]
```

In [ ]:

```
import geog0111.nasa_requests as nasa_requests
import numpy as np

# Loading user/pass combo as cylog failed repeatedly.
user, pas = np.loadtxt('/home/ucfatac/.userpass', dtype=str)


# Edited from Lewis' get_modis_tiles with assistance from Feng due to constant denial of access to
Nasa snow cover...
# site via previous methods.

for url in urls:
    with requests.Session() as session:
        # get password-authorised url
        session.auth = (user,pas)
        r1 = session.request('get',url)
        # this gets the url with codes for login etc.
        r2 = session.get(r1.url)
        # Split at start of filename "href="" with attention to correct data type.
        ret1 = [i.split('href="')[1] for i in r2.content.decode().split('\n') if ('MOD10A1' in i) &
('.hdf"><img' in i)]
                # After scanning through HTML data split at end of filename.
        fnames = [i.split('"><img ')[0] for i in ret1]
        # Create a link for the specfic file based on the above base_url and trimmed file names
        links = [url + i for i in fnames]

        for _,link in enumerate(links):
            # Enumerate through links looking for the specific relevant tile.
            # Write down to file in given directory.
            if tile in fnames[_]:
                r = session.get(link)
```

```
                with open('data/'+ fnames[_], 'wb') as fp:
                    r = fp.write(r.content)

# Files saved down to /data directory
```

In [ ]:

```
for file_name in filenames:
    # form full filename as a string
    # and print with an underline of =
    file_name = Path(file_name[0:4]).joinpath(file_name[5:]).as_posix()
    print(file_name)
    print('='*len(file_name))

    # open the file as g
    g = gdal.Open(file_name)
    # loop over the subdatasets
    for d in g.GetSubDatasets():
        print(d)
```

In [ ]:

```
from glob import glob

filenames = glob('data/MOD10A1*')


# Checking the data for the correct sub-dataset.


for file_name in filenames:
    # form full filename as a string
    # and print with an underline of =
    file_name = Path(file_name[0:4]).joinpath(file_name[5:]).as_posix()
    print(file_name)
    print('='*len(file_name))

    # open the file as g
    g = gdal.Open(file_name)
    # loop over the subdatasets
    for d in g.GetSubDatasets():
        print(d)
```

In [134]:

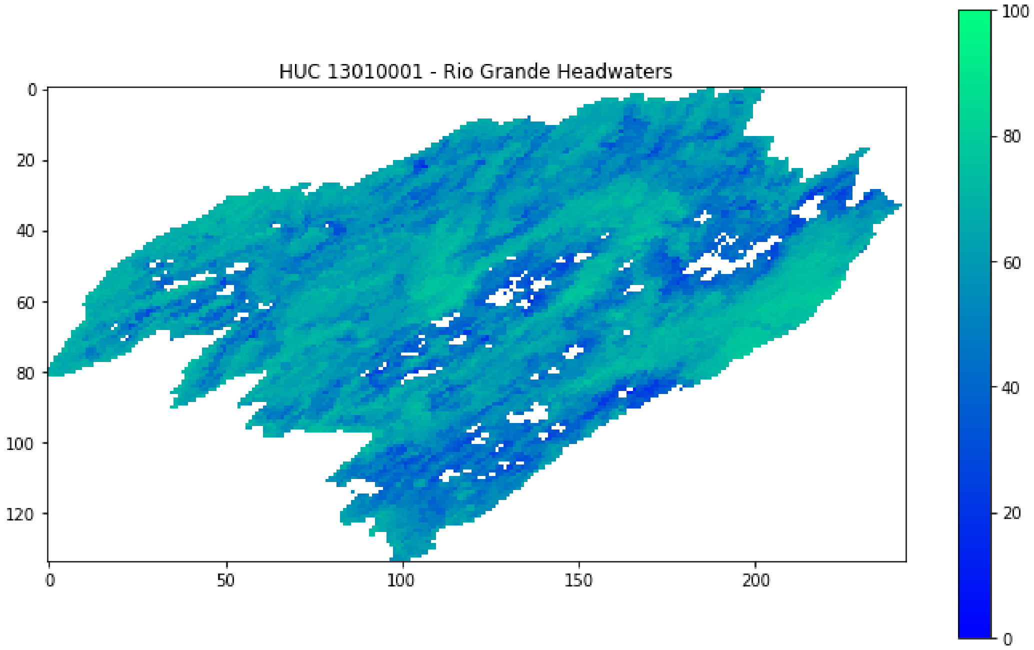
```
# Imported from 3.4 GDAL stacking and interpolating:

# Edited where appropriate to fir the demands of the data.

def find_files(year, doy,folder):
    """Find the files that have been downloaded.

    Keyword arguments:
    year -- The year of interest (int).
    doy -- The doy of interest (int).
    folder -- The directory for which to search for files (str).
    """
    data_folder = Path(folder)
    # Find all MOD files
    files = []
    sel_files = data_folder.glob(f"MOD10A1*.A{year:d}{doy:03d}.h09v05.*hdf")
    for fich in sel_files:
            files.append(fich)
    return files


def create_gdal_friendly_names(filenames, layer):

    """Form a filename compatible with GDAL.

    Keyword arguments:
    filenames -- The filenames to be found/converted (str).
```

```
    layer -- The sub-dataset layer of interest.
    """

    # Create GDAL friendly-names...
    gdal_filenames = []
    for file_name in filenames:
        fname = f'HDF4_EOS:EOS_GRID:'+\
                    f'"{file_name.as_posix()}":'+\
                    f'MOD_Grid_Snow_500m:{layer:s}'

        gdal_filenames.append(fname)
    return gdal_filenames



def clip(doy,
         year,
         tile = "h09v05",
         folder="data/",
         layer="NDSI_Snow_Cover",
         shpfile="data/Hydrological_Units/HUC_Polygons.shp",
         HUC="13010001",
         frmat = "MEM"):

    """
    Crop the data to the area of interest.

    Keyword arguments:
    tile -- The MODIS tile (str).
    doy -- The doy of interest (str).
    year -- The year of interest (int).
    folder -- The directory for which to search for files (str).
    layer -- The sub-dataset layer of interest (str).
    shpfile -- The directory/file used to crop data (str).
    HUC -- The Catchment (str).
    frmat -- The output format.
    """

    folder_path = Path(folder)
    # Find all files.
    hdf_files = find_files(year, doy, folder)

    # Create GDAL friendly-names...
    gdal_filenames = create_gdal_friendly_names(hdf_files, layer)

    if frmat == "MEM":
        g = gdal.Warp(
            "",
            gdal_filenames,
            format="MEM",
            dstNodata=255,
            cutlineDSName=shpfile,
            cutlineWhere="HUC=13010001",
            cropToCutline=True)

        data = g.ReadAsArray()
        # Adding a contingency to remove any values that do not relate to snow cover.
        data = data.astype("float")
        #data[data > 100] = np.nan
        return data


import matplotlib.pylab as plt
from pathlib import Path
import gdal

data = clip(doy = 1,
            year= 2016,)

# Adding a contingency to remove any values that do not relate to snow cover.
data = data.astype("float")
data[data > 100] = np.nan

plt.figure(figsize=(12, 12))
plt.title('HUC 13010001 - Rio Grande Headwaters')
plt.imshow(data, interpolation="nearest",vmin=0, vmax=100, cmap=plt.cm.winter)
```

```
plt.colorbar(shrink=0.6)
```

Out[134]:

<matplotlib.colorbar.Colorbar at 0x7efc3e2ec080>



Note the gaps inside the catchment where removed cloud data has obscured results.

Weighting Data:

Snow Cover & QA Data Descriptions

From: https://nsidc.org/data/MOD10A1

Scientific Data Set Description NDSI_Snow_Cover

NDSI snow cover plus other results. This value is computed for MOD10_L2 and retrieved when the observation of the day is selected. Possible values are:

```
0-100: NDSI snow cover
200: missing data
201: no decision
211: night
237: inland water
239: ocean
250: cloud
254: detector saturated
255: fill
```

Inland water should be replaced with a value of 0 for snow cover, assuming that when frozen the same pixel would return as snow cover. There may be a brief window of overlap where the water surface is frozen but doesn't have any snowcover, or does the sensor know the difference between frozen water and snow cover and still return a value of 237?

NDSI_Snow_Cover_Basic_QA

A basic estimate of the quality of the algorithm result. This value is computed for MOD10_L2 and retrieved with the corresponding observation of the day. Possible values are:

```
0: best
1: good
2: OK
3: poor (not currently in use)
```

```
211: night
239: ocean
255: unusable input or no data
```

The weights used will be as follows:

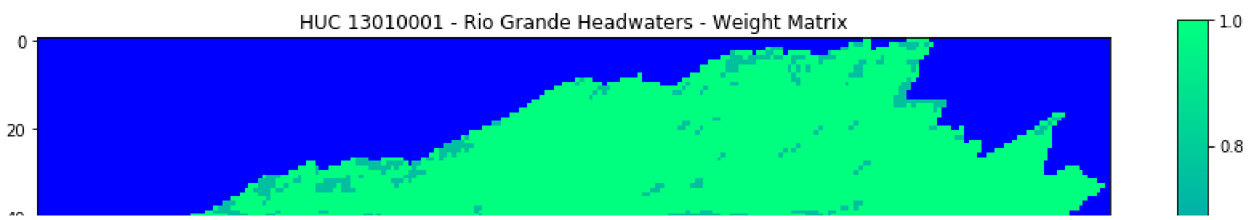
```
0: best, weight = 1
1: good, weight = 0.75
2: OK, weight = 0.5
3: poor (not currently in use), weight = 0.25
211: night, weight = 0
239: ocean, weight = 0 (Not applicable)
255: unusable input or no data, weight = 0
```

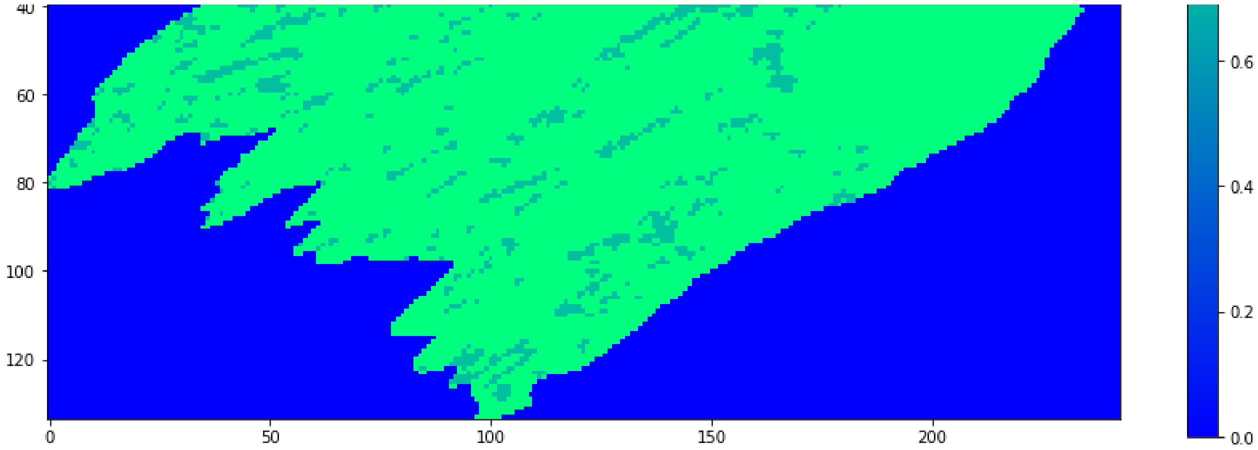
In [135]:

```python
qa = clip(1, 2016, layer="NDSI_Snow_Cover_Basic_QA")

# Copied from notes chapter 3.4,

def get_scaling(qa):

    """
    Creates a weight matrix, based upon an input array of QA values.

    Keyword arguments:

    qa = Input array (array)

    """

    # Creates an array the same size as the input:
    weight = np.zeros_like(qa, dtype=np.float)
    # Create a weight loop to assign weights to new array based on QA indicators.
    for qa_val in [0, 1, 2, 3, 211, 239, 255]:
        if qa_val == 0:
            weight[qa == qa_val] = 1
        if qa_val == 1:
            weight[qa == qa_val] = 0.75
        if qa_val == 2:
            weight[qa == qa_val] = 0.5
        if qa_val == 3:
            weight[qa == qa_val] = 0.25
        # Sets all useless values to zero, uneeded but for posterity.
        if qa_val == 211 or qa_val == 239 or qa_val == 255:
            weight[qa == qa_val] = 0
    weight = weight.astype("float")
    return weight

# Plot the weight matrix as a test to see the distribution of QA throughout the catchment:

weights = get_scaling(qa)

weights = weights.astype("float")

plt.figure(figsize=(15, 12))
plt.title('HUC 13010001 - Rio Grande Headwaters - Weight Matrix')
plt.imshow(weights, interpolation="nearest",vmin=0, vmax=1, cmap=plt.cm.winter)

plt.colorbar(shrink=0.6)
```

Out[135]:

```
<matplotlib.colorbar.Colorbar at 0x7efc3e2c4a20>
```





In [136]:

```
# A function to return a weight matrix and snow cover for a given date.
# Imported and edited from notes chapter 3.4

def process_single_date(doy,
                        year,
                        tile = "h09v05",
                        folder="data/",
                        shpfile="data/Hydrological_Units/HUC_Polygons.shp",
                        HUC="13010001"):
    """
    Uses the established clip and scaling functions to return data and weight arrays.

    Keyword arguments:

    doy = Day of year (int)
    year = Year (int)
    tile = Modis tile in question (str)
    folder = Local directory of file (str)
    shpfile = File directory of shape file (str)
    HUC = Catchment code (str)

    """

    snow_data = clip(doy,
                        year,
                        tile = "h09v05",
                        folder=folder,
                        layer="NDSI_Snow_Cover",
                        shpfile=shpfile)

    qa_data = clip(doy,
                      year,
                      tile = "h09v05",
                      folder=folder,
                      layer="NDSI_Snow_Cover_Basic_QA",
                      shpfile=shpfile)


    weights = get_scaling(qa_data)

    return snow_data, weights
```

Final Data Clipping as Timeseries:

In [ ]:

```
from datetime import datetime, timedelta

# A function to return a weight matrix and snow cover for a given time-series.
# Imported and edited from notes chapter 3.4

def process_timeseries(year,
                           tile = "h09v05",
                           folder="data/"
```

```
                        folder="data/",
                        shpfile="data/Hydrological_Units/HUC_Polygons.shp",
                        HUC="13010001",
                        verbose=True):

    """
    Outputs a cropped data and weight matrix for a specified year.

    Keyword arguments:

    year = Year (int)
    tile = MODIS tile (str)
    folder = Directory of data (str)
    shpfile = File directory of shapefile (str)
    HUC = Catchment code (str)
    Verbose = Used to output logs (bool)

    """

    today = datetime(year, 1, 1)
    # length = len(datetime(year))
    dates = []
    # Creating a check to find the length of years depending on whether or not said year is a leap
year.
    if year % 4 == 0:
        rang = range(366)
        ran = 366
    if year % 4 != 0:
        rang = range(365)
        ran = 365
    # Now function is adaptable to leap years:
    for i in rang:
        if (i%1 == 0) and verbose:
            print(f"Doing {str(today):s}")
        if today.year != year:
            break
        doy = int(today.strftime("%j"))
        # Add a try/except condition to catch missing days where satellite did not scan.
        try:
            this_snow, this_weight = process_single_date(
                doy,
                year,
                tile=tile,
                folder="data/",
                shpfile="data/Hydrological_Units/HUC_Polygons.shp",
                HUC="13010001")
        # Print an error message:
        except(AttributeError):
            print("Attribute error encountered")
        if doy == 1:
            # Create outputs!

            ny, nx = this_snow.shape
            snow_array = np.zeros((ny, nx, ran))
            weights_array = np.zeros((ny, nx, ran))
        snow_array[:, :, i] = this_snow
        weights_array[:, :, i] = this_weight
        dates.append(today)
        today = today + timedelta(days=1)
    return dates, snow_array, weights_array

dates16, snow_2016, weight_2016 = process_timeseries(2016)
dates17, snow_2017, weight_2017 = process_timeseries(2017)
```

Attending to Missing Values in Data:

In [283]:

```
#200: missing data
#201: no decision
#211: night
#237: inland water
#239: ocean
#250: cloud
#254: detector saturated
```

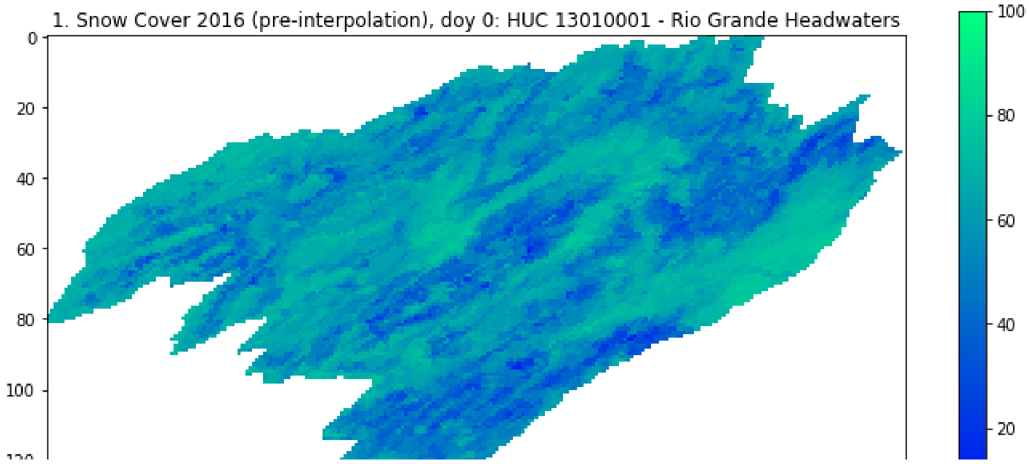
```
#201: detector saturated
#255: fill


def fix_missing(snow_cover, weight):

    """
    Replaces missing values in snow cover arrays with previous days by carrying forward data.

    Keyword arguments:

    snow_cover = Input snow cover array (array)
    weight = Input weight array (array)

    """

    # Adjusting weight matrix for missing values in snow data.
    weight[200 <= snow_cover] = 0
    # Start by creating a blank array equal to the size of the snow cover array.
    new_cover = np.zeros_like(snow_cover, dtype=np.float)

    new_weight = weight
    # Converting inland water (237) into snow cover 0.
    snow_cover[snow_cover == 237] = 0
    # Creating a variable check for leap years.
    if snow_cover.shape[2] % 4 == 0:
        rang = range(366)
        ran = 366
    if snow_cover.shape[2] % 4 != 0:
        rang = range(365)
        ran = 365
    # Iterate through each pixel of the array.
    for x in range(snow_cover.shape[0]):
        for y in range(snow_cover.shape[1]):
            for n in range(snow_cover.shape[2]):
                # if pixel value is a valid snow cover number write to new array
                if 0 <= snow_cover[x, y, n] <= 100:
                    for i in range(n, ran):
                    # write pixel to every day of the array (going forward only), so that successive
                    # days where data is missing are covered by those that went before.
                        new_cover[x, y, i] = snow_cover[x, y, n]
                        new_weight[x, y, i] = weight[x, y, n]
                # Keep the 255/fill values as they are weighted to 0 and only surround the cropped image.
                if snow_cover[x, y, n] == 255:
                    for i in range(n, ran):
                        new_cover[x, y, i] = snow_cover[x, y, n]
                        new_weight[x, y, i] = weight[x, y, n]
                # Pass over invalid values - not written to new array
                if snow_cover[x, y, n] == 200:
                    pass
                if snow_cover[x, y, n] == 201:
                    pass
                if snow_cover[x, y, n] == 250:
                    pass
    # Relooping from the start to fix missing values passed above.
    # The first day in any year is susceptible as there as no values being written forward
    # In this case use the same methods to work backwards.
    for x in range(snow_cover.shape[0]):
        for y in range(snow_cover.shape[1]):
            for n in range(snow_cover.shape[2]):
                if snow_cover[x, y, n] == 200:
                    for i in range(n, ran):
                        if snow_cover[x, y, i] == 200:
                            continue
                        if snow_cover[x, y, i] == 201:
                            continue
                        if snow_cover[x, y, i] == 250:
                            continue
                        if snow_cover[x, y, i] == 255:
                            continue
                        if snow_cover[x, y, i] <= 100:
                            holder = snow_cover[x, y, i]
                            new_cover[x, y, n] = holder
                            break
                if snow_cover[x, y, n] == 201:
                    for i in range(n, ran):
```

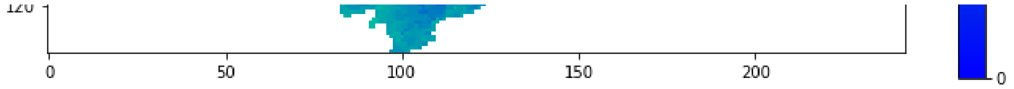
```
                    for i in range(n, ran):
                        if snow_cover[x, y, i] == 200:
                            continue
                        if snow_cover[x, y, i] == 201:
                            continue
                        if snow_cover[x, y, i] == 250:
                            continue
                        if snow_cover[x, y, i] == 255:
                            continue
                        if snow_cover[x, y, i] <= 100:
                            holder = snow_cover[x, y, i]
                            new_cover[x, y, n] = holder
                            break
                if snow_cover[x, y, n] == 250:
                    for i in range(n, ran):
                        if snow_cover[x, y, i] == 200:
                            continue
                        if snow_cover[x, y, i] == 201:
                            continue
                        if snow_cover[x, y, i] == 250:
                            continue
                        if snow_cover[x, y, i] == 255:
                            continue
                        if snow_cover[x, y, i] <= 100:
                            holder = snow_cover[x, y, i]
                            new_cover[x, y, n] = holder
                            break


    return new_cover, new_weight

# Get new snow and weight arrays using above function.

cover_2016, nweight_2016 = fix_missing(snow_2016, weight_2016)
cover_2017, nweight_2017 = fix_missing(snow_2017, weight_2017)
```

Spatial Representation of Snow Cover Flux: 2016

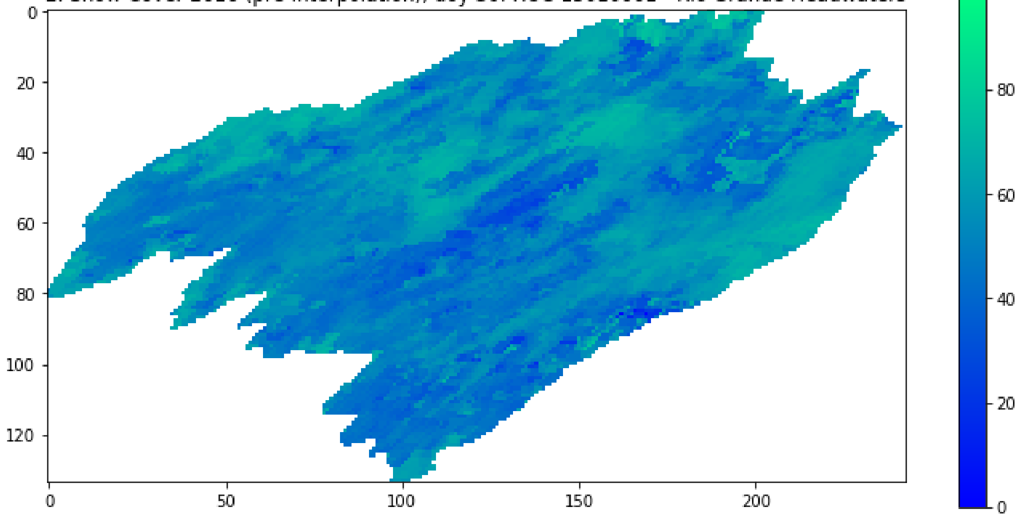
In [286]:

```
# Plot 13 evenly spaced days of data for 2016.

x = range(0, 366, 30)

# Removed fill values for better understanding of catchment borders without confusing 255 value sc
aling.
cover_2016 = cover_2016.astype("float")
cover_2016[cover_2016 > 100] = np.nan

for i in enumerate(x):
    plt.figure(figsize=(12, 12))
    plt.title(f'{i[0]+1}. Snow Cover 2016 (pre-interpolation), doy {i[1]}: HUC 13010001 - Rio
Grande Headwaters')
    plt.imshow(cover_2016[...,i[1]], interpolation="nearest",vmin=0, vmax=100, cmap=plt.cm.winter)

    plt.colorbar(shrink=0.5)
```

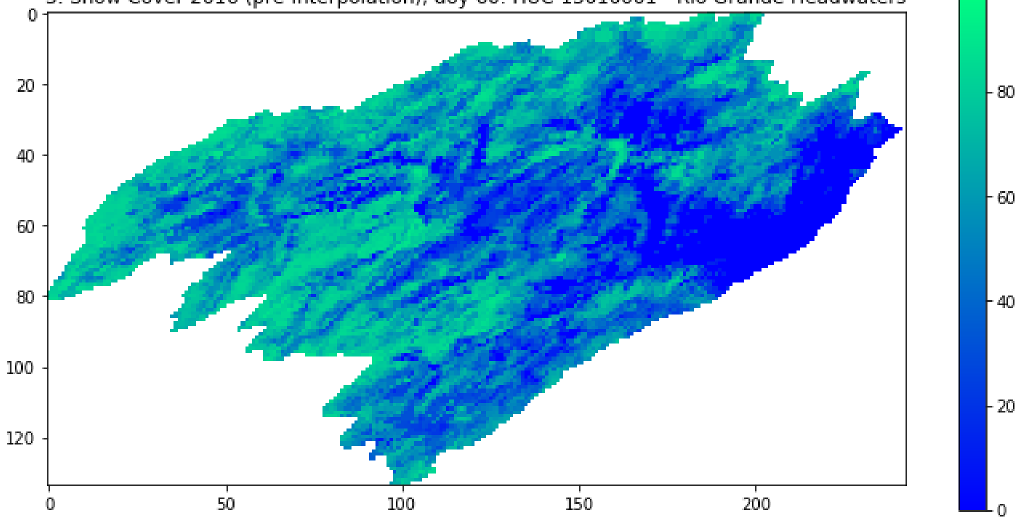




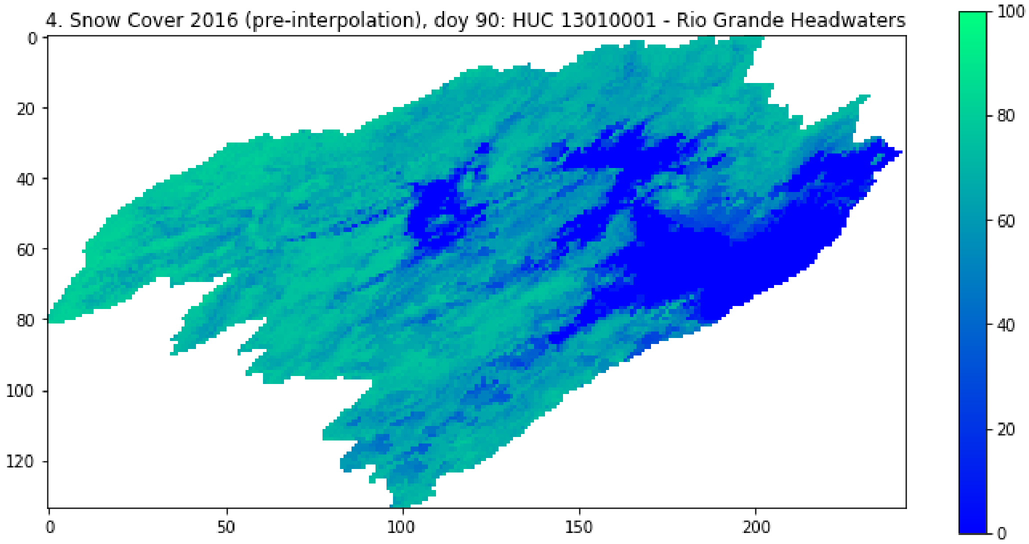
2. Snow Cover 2016 (pre-interpolation), doy 30: HUC 13010001 - Rio Grande Headwaters



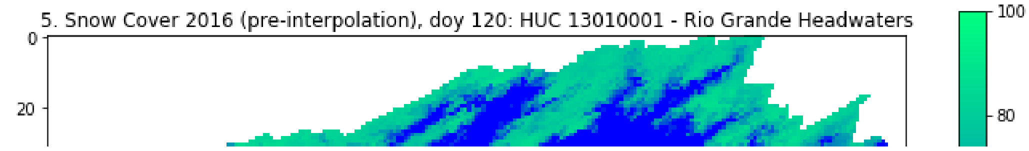
3. Snow Cover 2016 (pre-interpolation), doy 60: HUC 13010001 - Rio Grande Headwaters

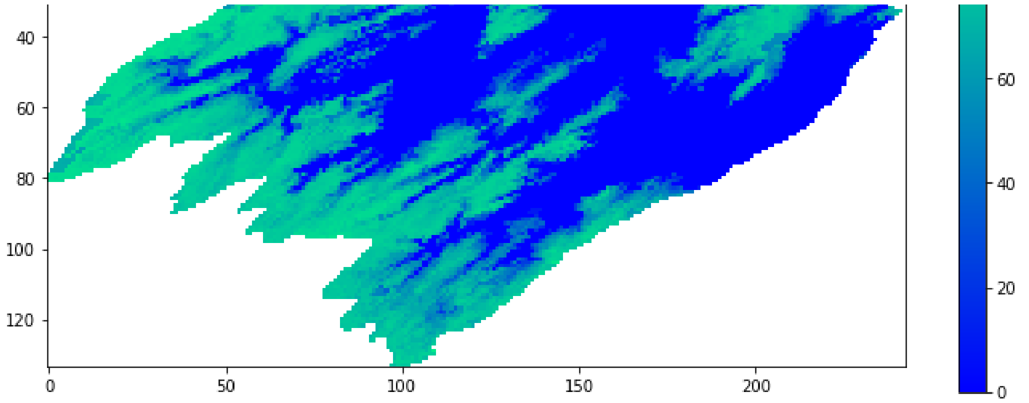


4. Snow Cover 2016 (pre-interpolation), doy 90: HUC 13010001 - Rio Grande Headwaters

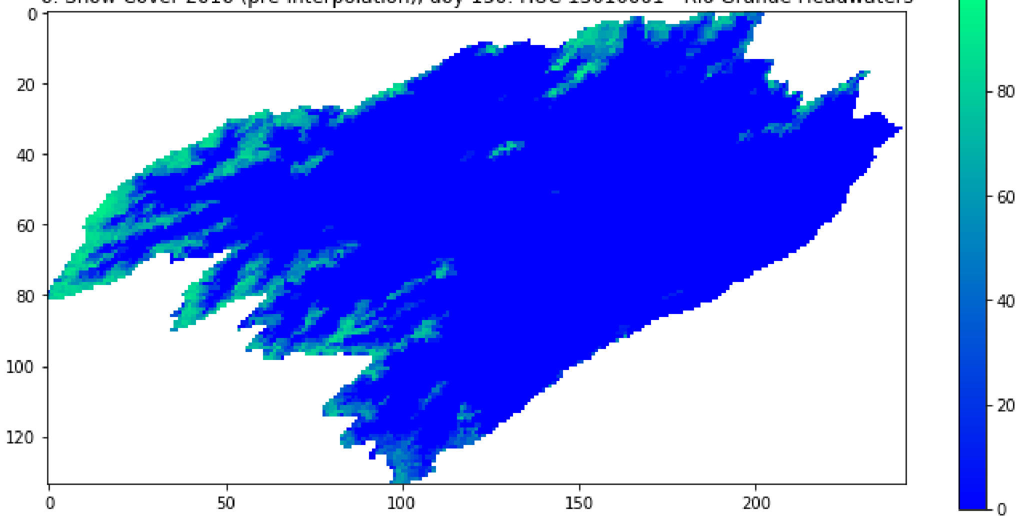


5. Snow Cover 2016 (pre-interpolation), doy 120: HUC 13010001 - Rio Grande Headwaters

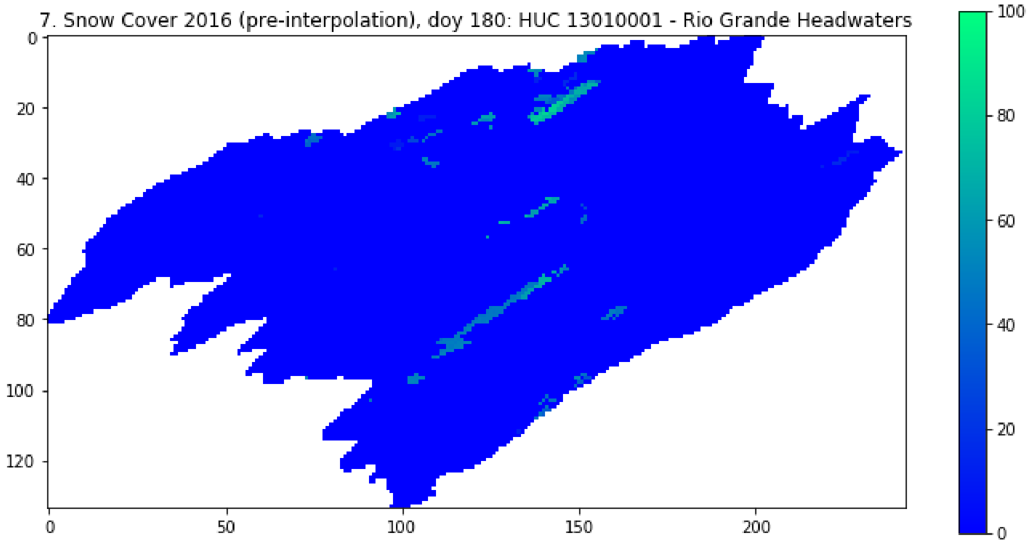




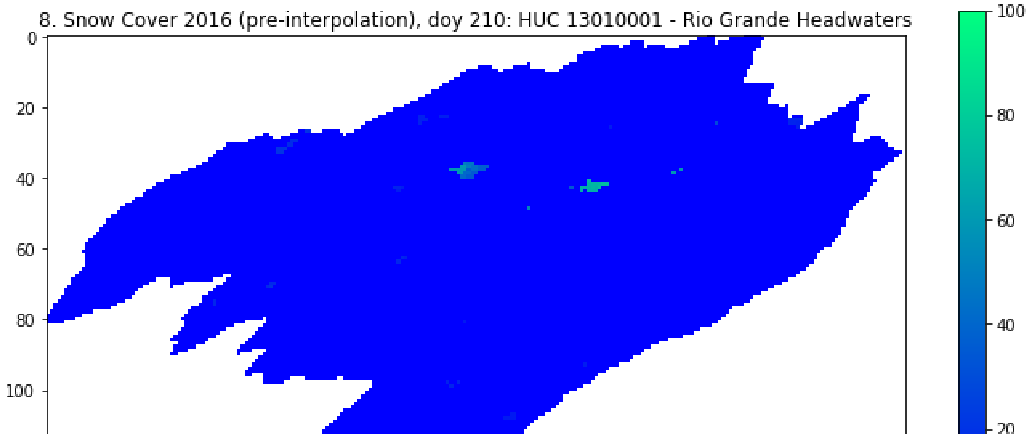
6. Snow Cover 2016 (pre-interpolation), doy 150: HUC 13010001 - Rio Grande Headwaters

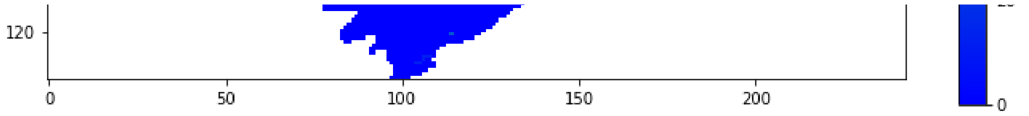


7. Snow Cover 2016 (pre-interpolation), doy 180: HUC 13010001 - Rio Grande Headwaters

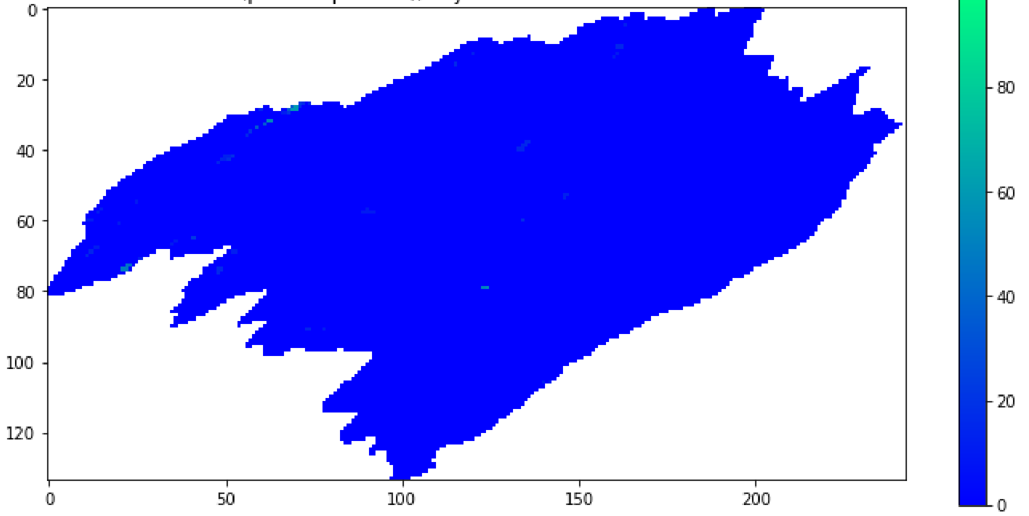


8. Snow Cover 2016 (pre-interpolation), doy 210: HUC 13010001 - Rio Grande Headwaters

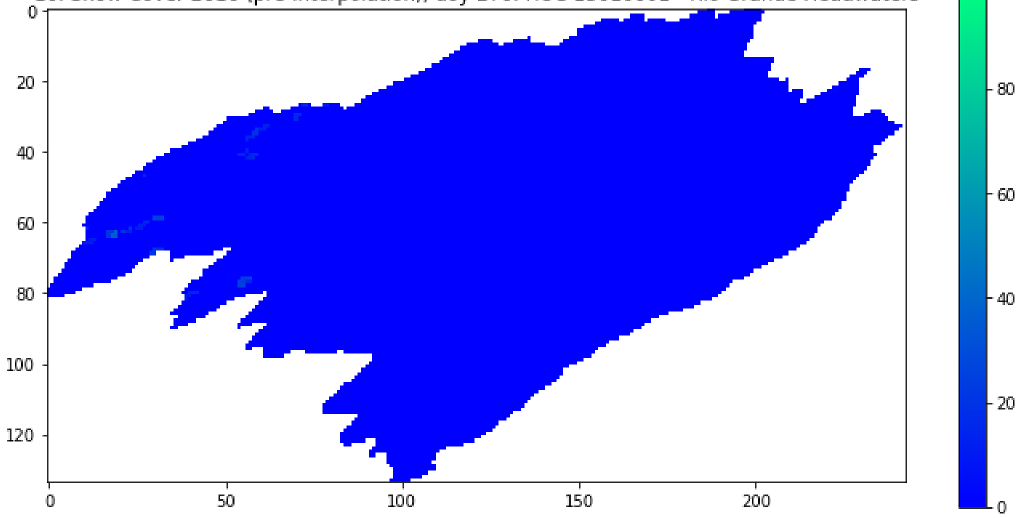




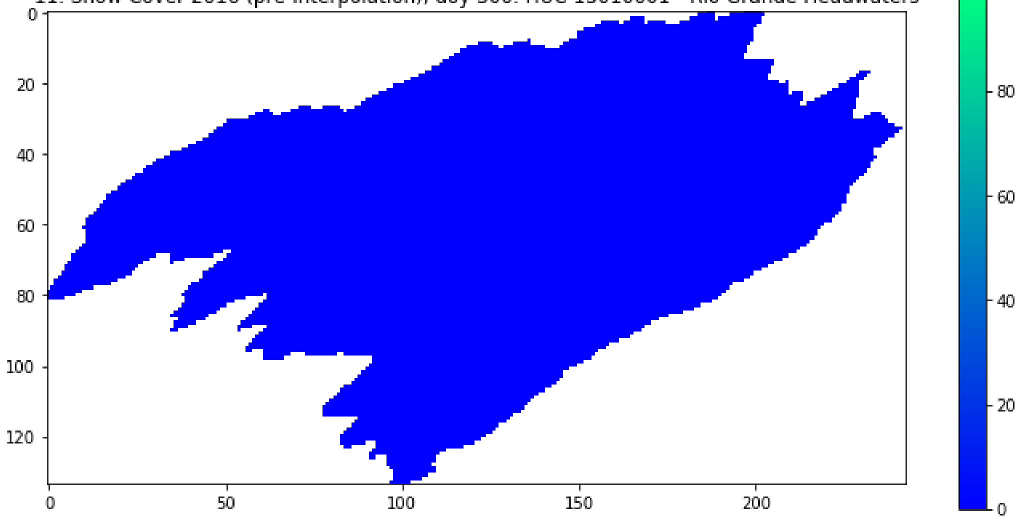
9. Snow Cover 2016 (pre-interpolation), doy 240: HUC 13010001 - Rio Grande Headwaters



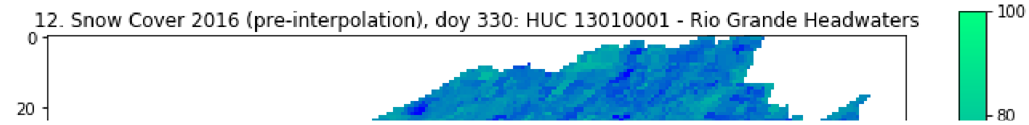
10. Snow Cover 2016 (pre-interpolation), doy 270: HUC 13010001 - Rio Grande Headwaters

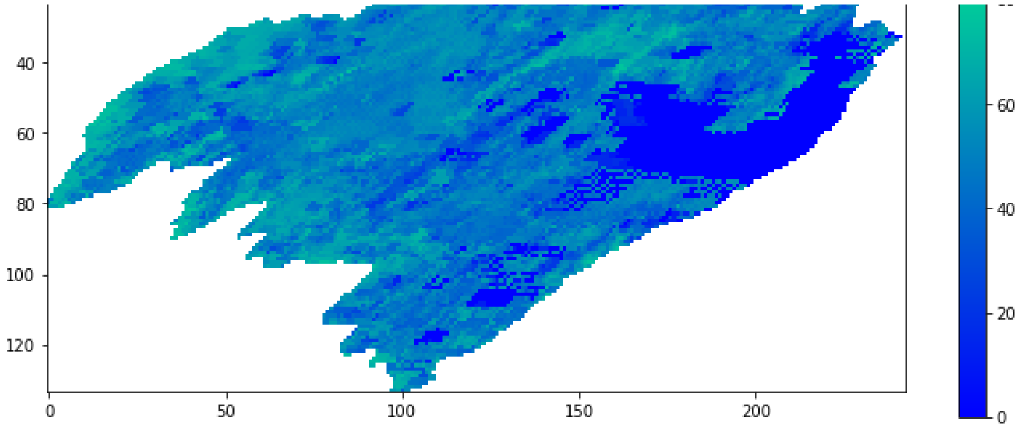


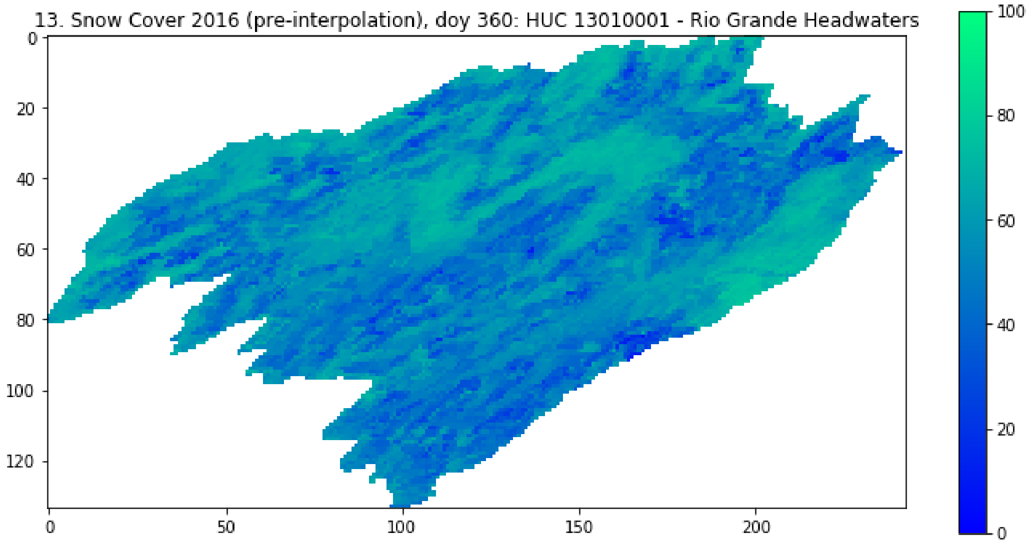
11. Snow Cover 2016 (pre-interpolation), doy 300: HUC 13010001 - Rio Grande Headwaters



12. Snow Cover 2016 (pre-interpolation), doy 330: HUC 13010001 - Rio Grande Headwaters



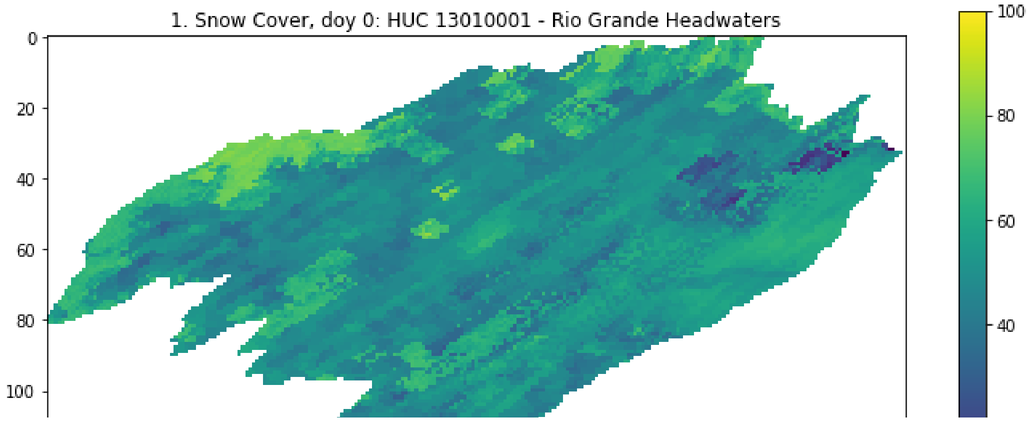


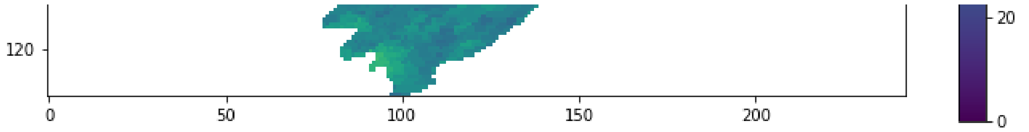


Spatial Representation of Snow Cover Flux: 2017

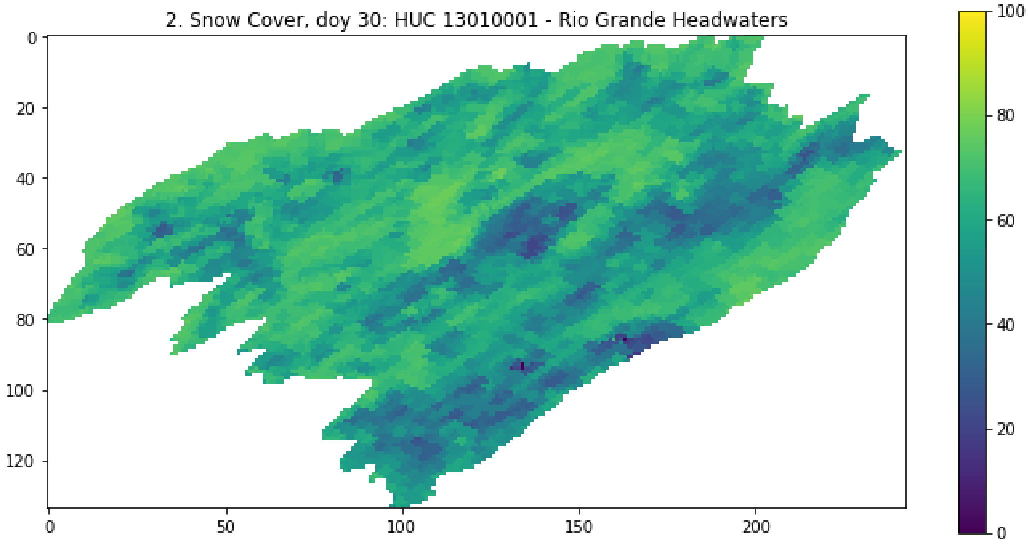
In [287]:

```
# Plot 13 evenly spaced days of data for 2016.
# Used different colour scheme for visual distinction between each year.

x = range(0, 365, 30)

cover_2017 = cover_2017.astype("float")
cover_2017[cover_2017 > 100] = np.nan

for i in enumerate(x):
    plt.figure(figsize=(12, 12))
    plt.title(f'{i[0]+1}. Snow Cover, doy {i[1]}: HUC 13010001 - Rio Grande Headwaters')
    plt.imshow(cover_2017[...,i[1]], interpolation="nearest",vmin=0, vmax=100, cmap=plt.cm.viridis)

    plt.colorbar(shrink=0.5)
```

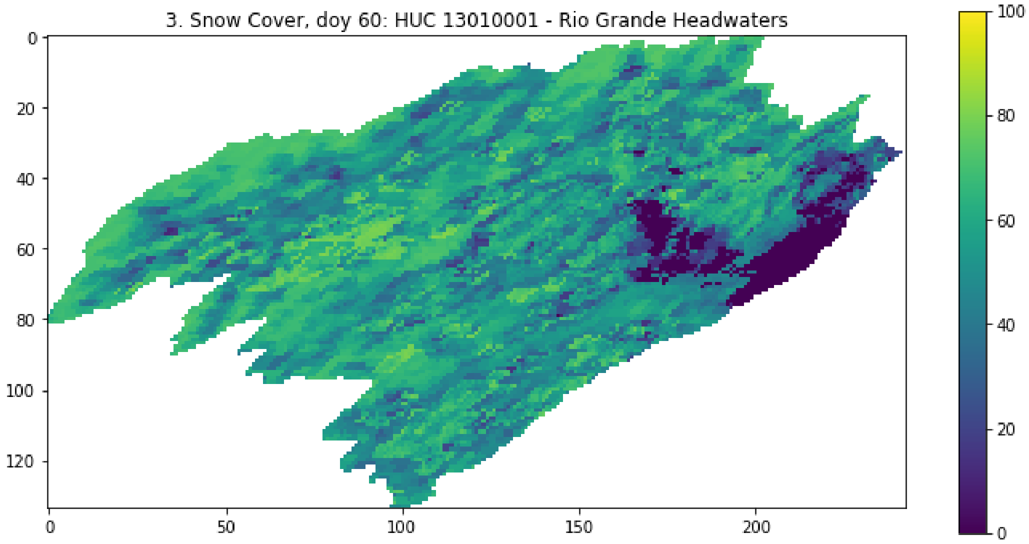




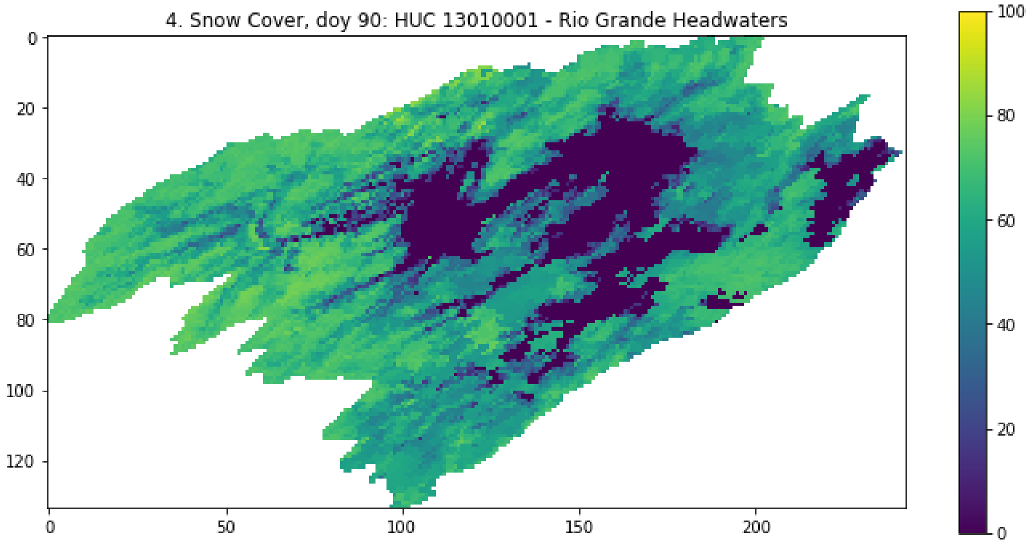
2. Snow Cover, doy 30: HUC 13010001 - Rio Grande Headwaters



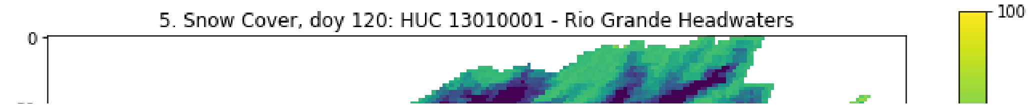
3. Snow Cover, doy 60: HUC 13010001 - Rio Grande Headwaters

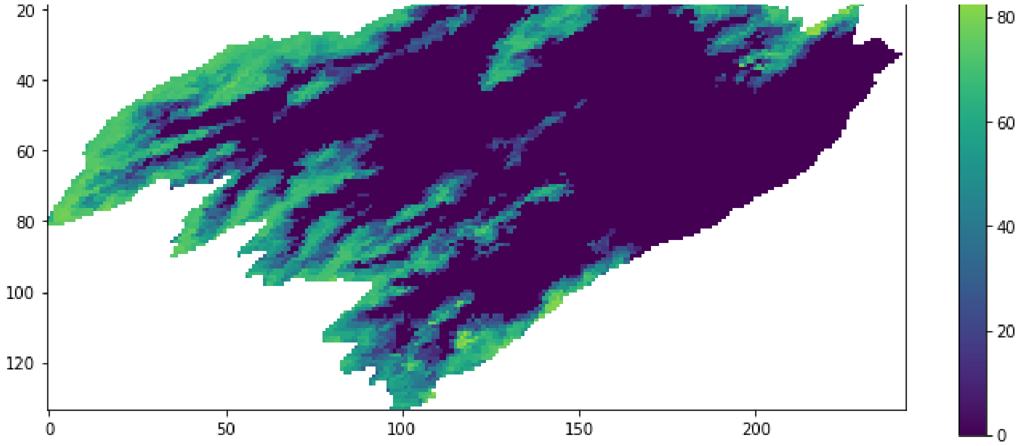


4. Snow Cover, doy 90: HUC 13010001 - Rio Grande Headwaters

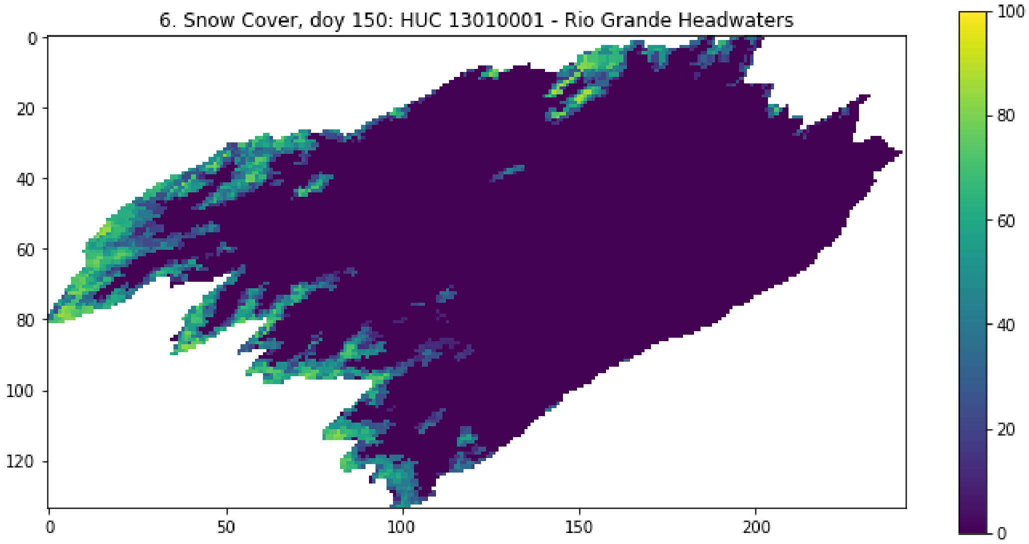


5. Snow Cover, doy 120: HUC 13010001 - Rio Grande Headwaters

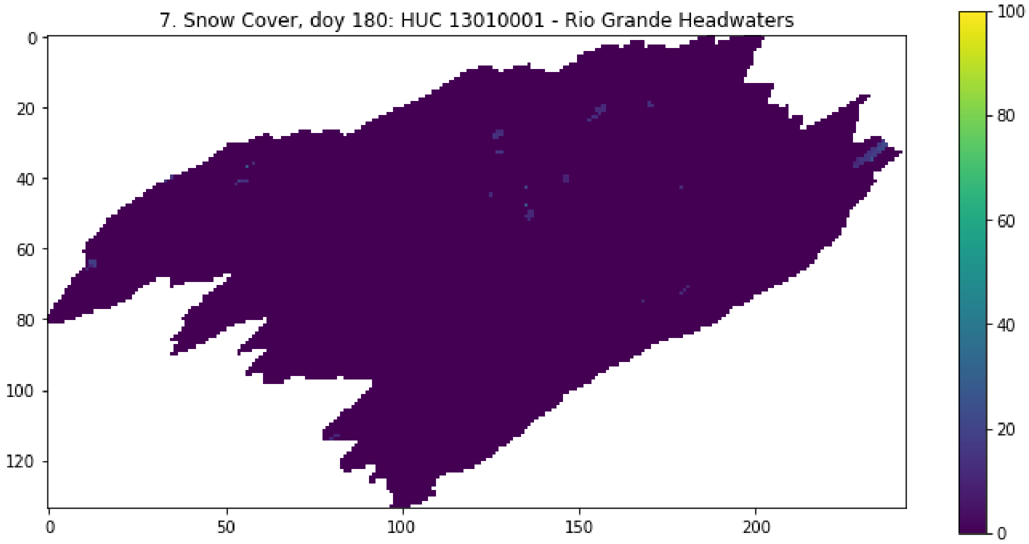




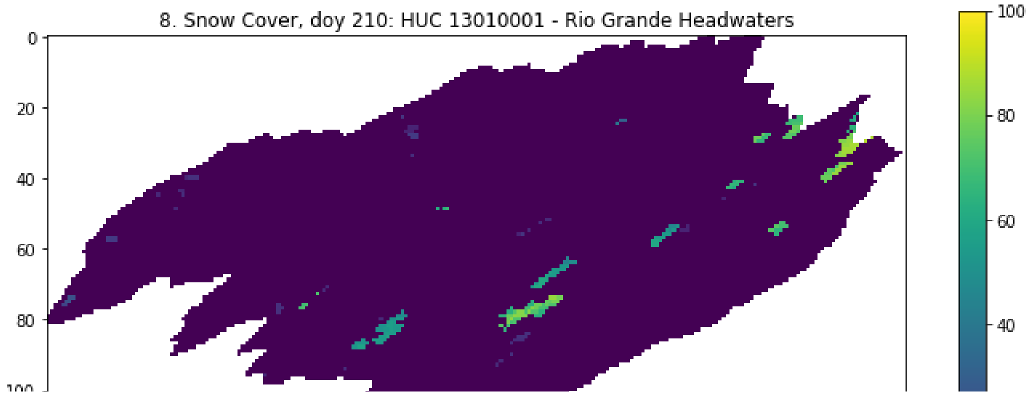
6. Snow Cover, doy 150: HUC 13010001 - Rio Grande Headwaters

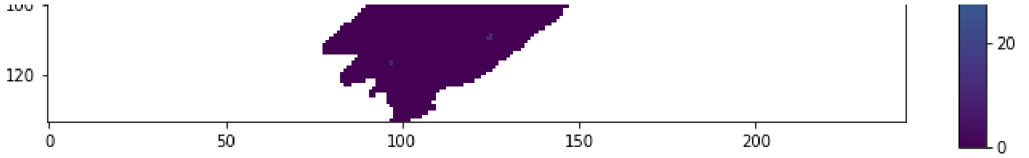


7. Snow Cover, doy 180: HUC 13010001 - Rio Grande Headwaters

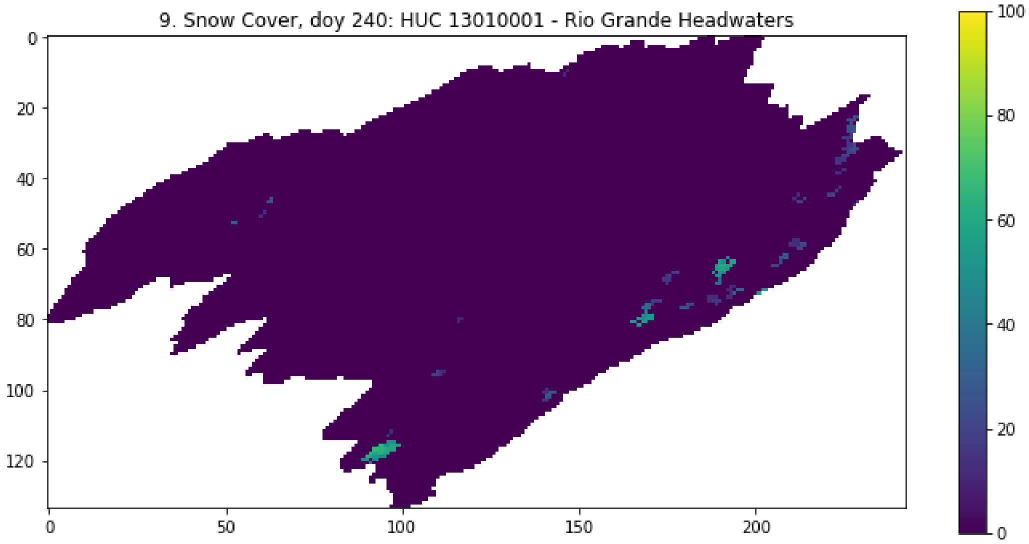


8. Snow Cover, doy 210: HUC 13010001 - Rio Grande Headwaters

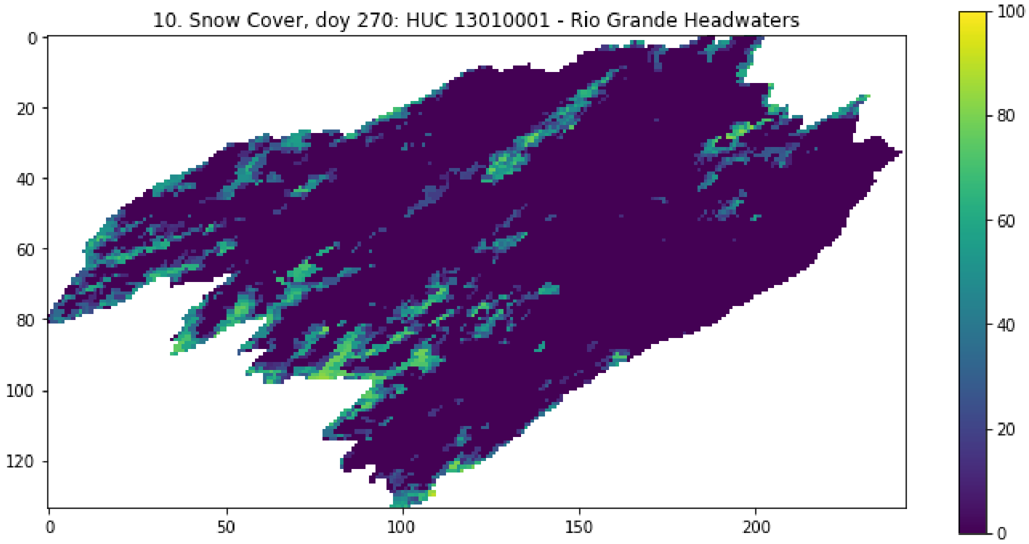




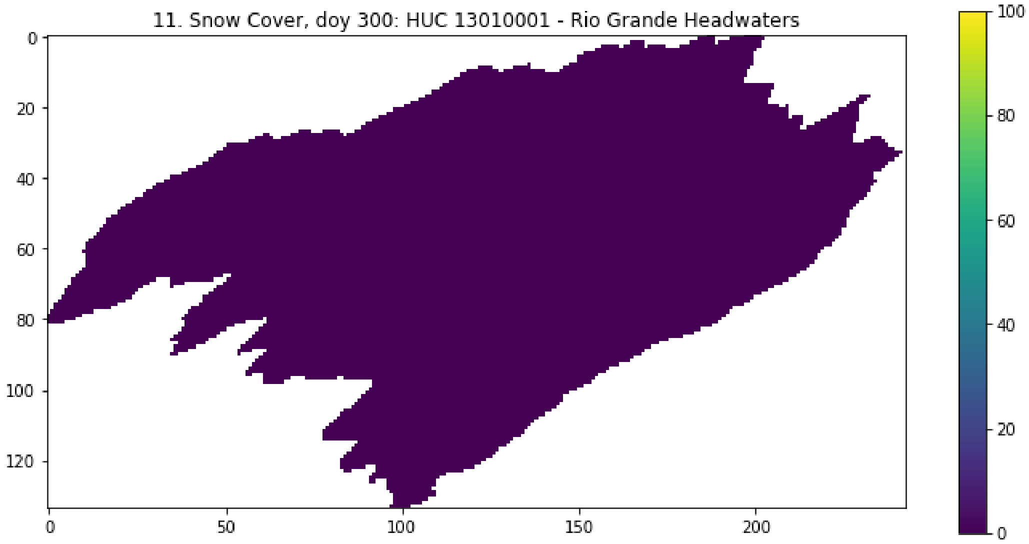
9. Snow Cover, doy 240: HUC 13010001 - Rio Grande Headwaters



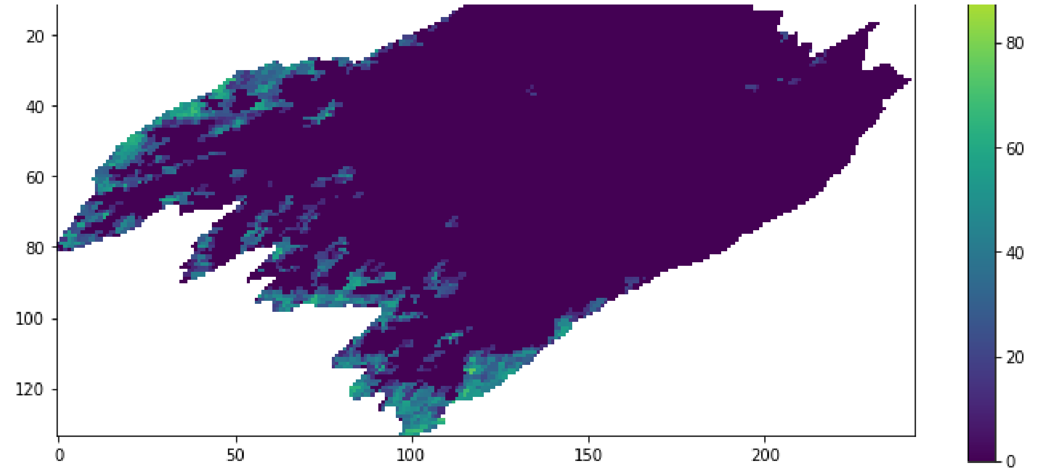
10. Snow Cover, doy 270: HUC 13010001 - Rio Grande Headwaters

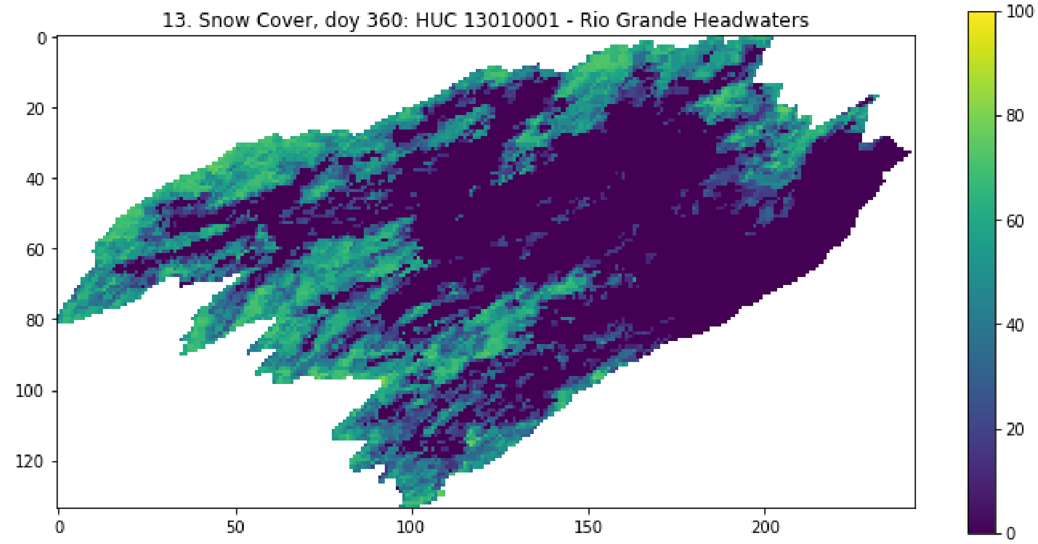


11. Snow Cover, doy 300: HUC 13010001 - Rio Grande Headwaters



12. Snow Cover, doy 330: HUC 13010001 - Rio Grande Headwaters





Analysis

For both years there exist regions within the catchment which seem to be longer lasting in their coverage than others, it is likely that both altitude and daily shading play a role here but without the tools to further investigate this must remain an assumption.

The catchment is completely clear of snow during the summer in both years and doesn't begin to recover full coverage until the last 60 days of the year. It is hard to make out much difference in cover which could be behind the difference in meltwater/discharge flux noted between the years in section 1. There does appear to be a slight difference between the years in that 2017 sees a slight reduction in cover and sees an early and more substantial spring melt.

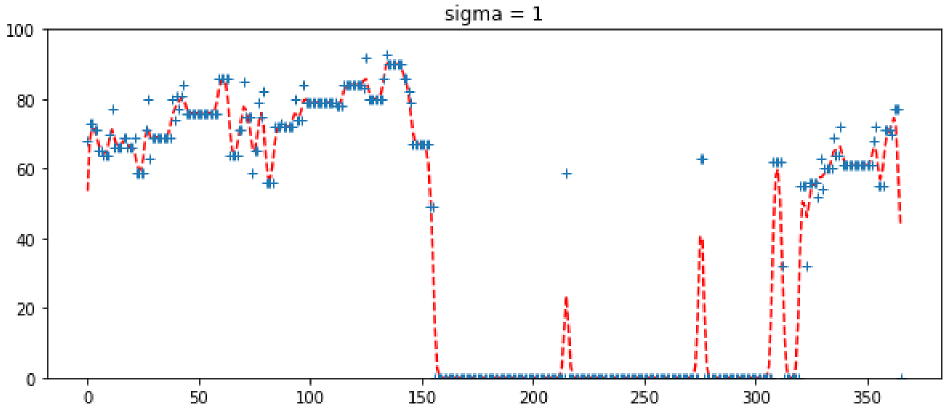
Weighted Interpolation

In [288]:

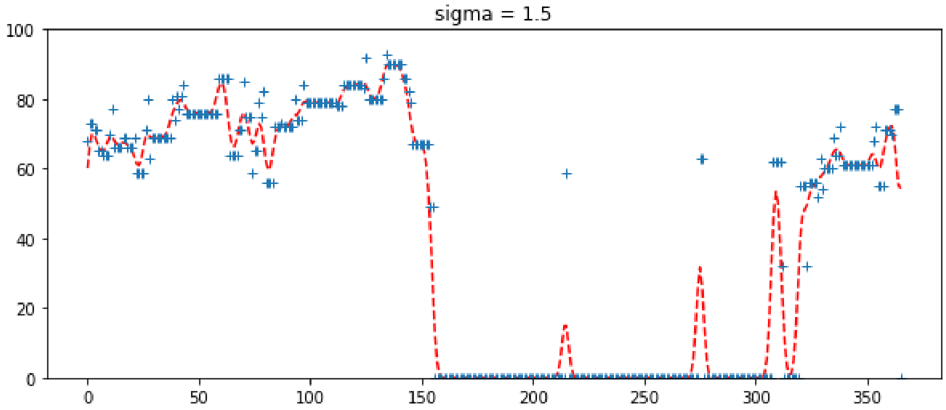
```
import scipy
import scipy.ndimage.filters

# testing sigma values for best fit.
sigma = [1, 1.5, 2, 2.5, 3, 4, 6, 7, 8,]

# basis of code taken from exercise 3.4:
for i in sigma:
    x = np.arange(-3*i,3*i+1)
    gaussian = np.exp((-(x/i)**2)/2.0)

    numerator = scipy.ndimage.filters.convolve1d(cover_2016 * nweight_2016, gaussian, axis=2,mode='
wrap')

    denominator = scipy.ndimage.filters.convolve1d(nweight_2016, gaussian, axis=2,mode='wrap')

    # avoid divide by 0 problems by setting zero values
```

```
    # of the denominator to not a number (NaN)
    denominator[denominator==0] = np.nan

    interpolated_snow = numerator/denominator

    sweight = nweight_2016.sum(axis=2)
    r,c = np.where(sweight == np.max(sweight))
    plt.figure(figsize=(10,4))

    ipixel = 0 # To plot the i-th pixel
    plt.plot((interpolated_snow)[r[ipixel],c[ipixel],:],'r--')
    plt.plot((cover_2016)[r[ipixel],c[ipixel],:],'+')
    plt.title(f'sigma = {i}')
    plt.ylim(0,100)
```

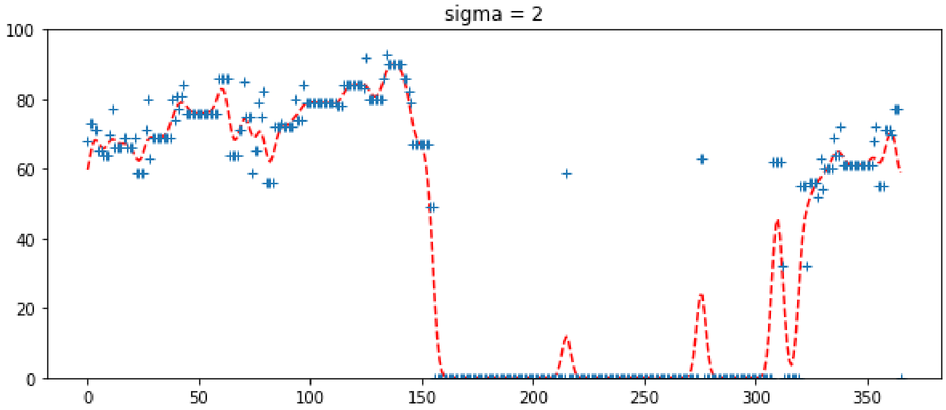
sigma = 1



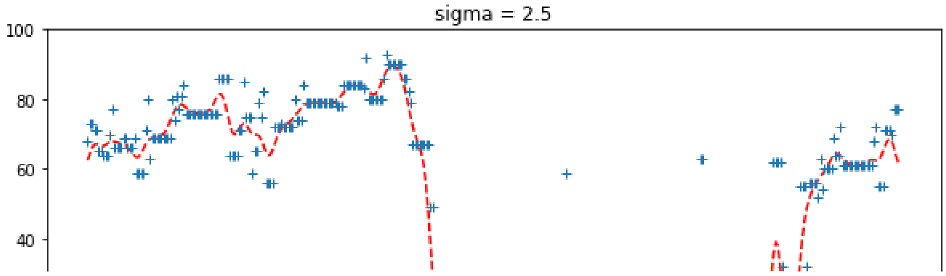
sigma = 1.5

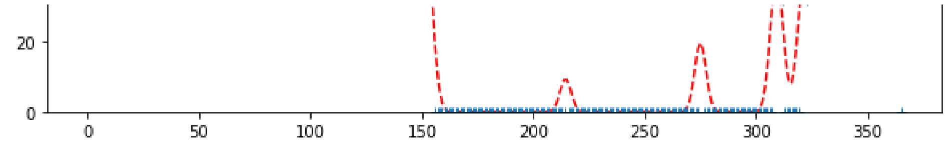


sigma = 2

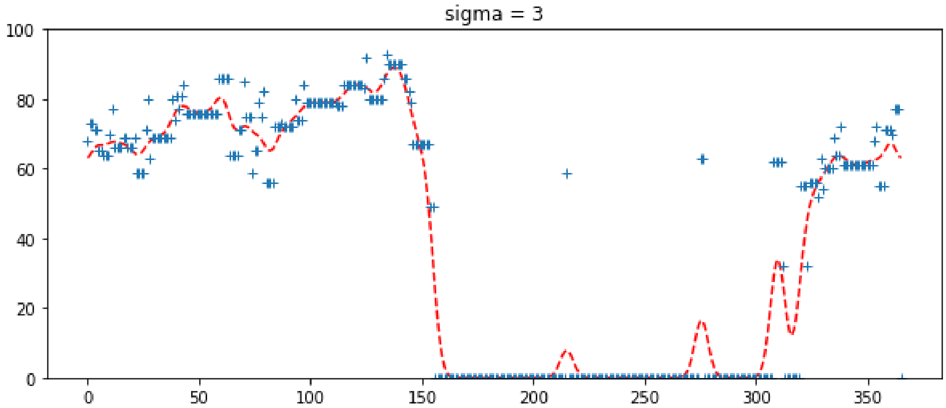


sigma = 2.5

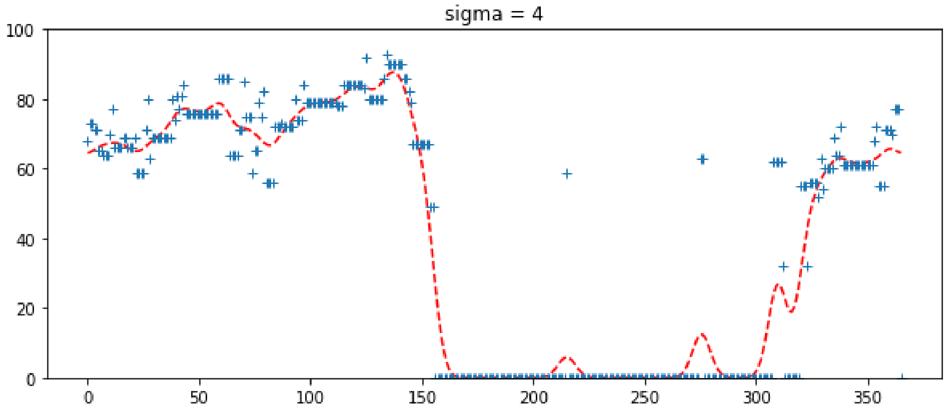




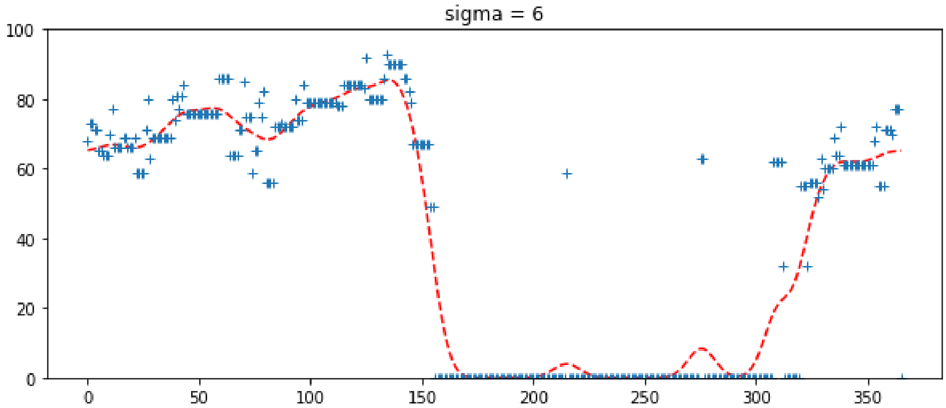
sigma = 3



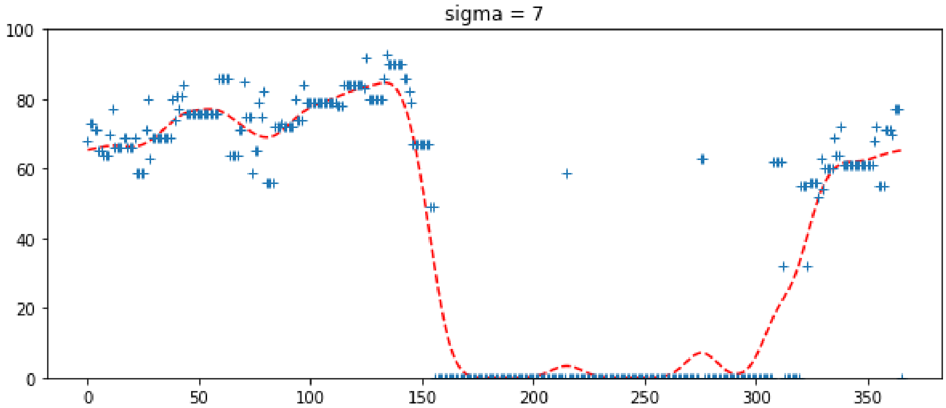
sigma = 4



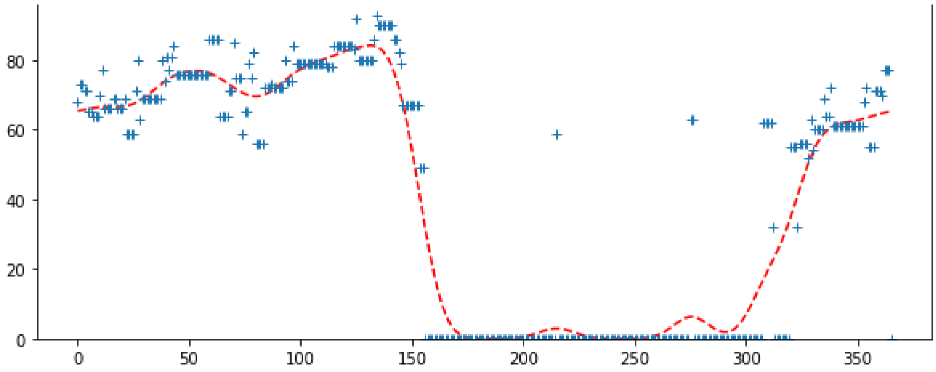
sigma = 6



sigma = 7



sigma = 8



After some experimentation, the value of sigma which seems to map the snow cover flux most accurately seems to be in the region of 1.5. Snow melt and fall are relatively short events so a sigma value that is higher does not map well to data where snow cover is quickly changing. Sigma 1.5

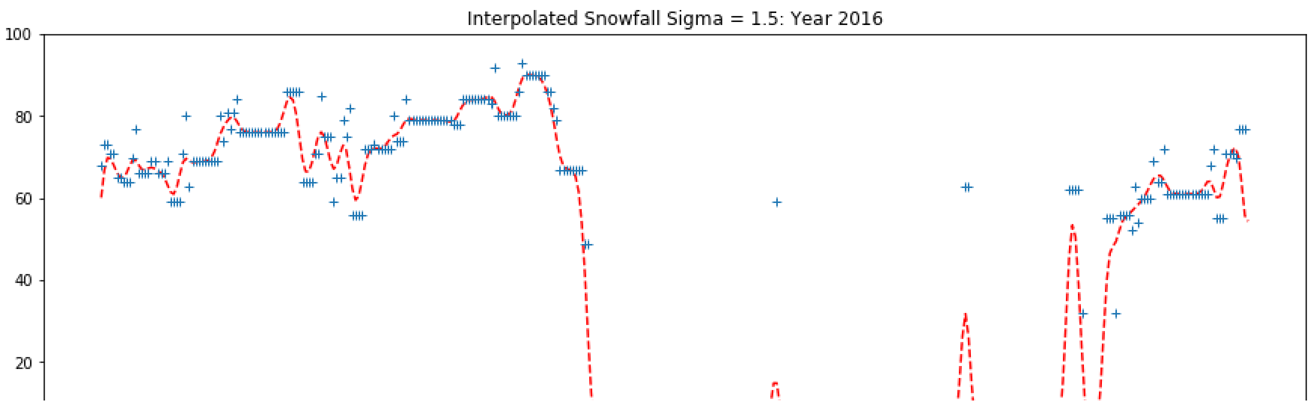
Interpolated Snow Cover 2016

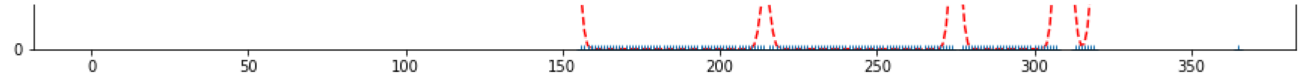
In [289]:

```
# Plotting interpolated graphs of snow cover for 2016

sigma = 1.5

x = np.arange(-3*sigma,3*sigma+1)
gaussian = np.exp((-(x/sigma)**2)/2.0)

numerator = scipy.ndimage.filters.convolve1d(cover_2016 * nweight_2016, gaussian, axis=2,mode='wrap
')

denominator = scipy.ndimage.filters.convolve1d(nweight_2016, gaussian, axis=2,mode='wrap')

# avoid divide by 0 problems by setting zero values
# of the denominator to not a number (NaN)
denominator[denominator==0] = np.nan

interpolated_snow_16 = numerator/denominator


sweight = nweight_2016.sum(axis=2)
r,c = np.where(sweight == np.max(sweight))
plt.figure(figsize=(15,5))

#snow_2016[snow_2016 > 100] = np.nan

ipixel = 0 # To plot the i-th pixel
plt.plot((interpolated_snow_16)[r[ipixel],c[ipixel],:],'r--')
plt.plot((cover_2016)[r[ipixel],c[ipixel],:],'+')
plt.title('Interpolated Snowfall Sigma = 1.5: Year 2016')
plt.ylim(0,100)
```

Out[289]:

```
(0, 100)
```





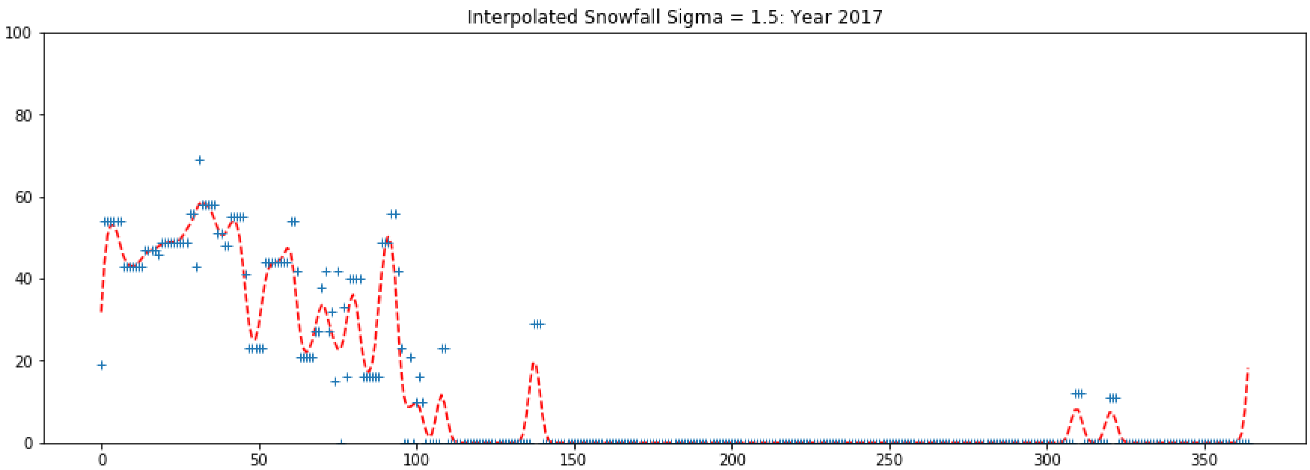
Interpolated Snow Cover 2017

In [290]:

```
# Plotting interpolated graphs of snow cover for 2017

sigma = 1.5

x = np.arange(-3*sigma,3*sigma+1)
gaussian = np.exp((-(x/sigma)**2)/2.0)

numerator = scipy.ndimage.filters.convolve1d(cover_2017 * nweight_2017, gaussian, axis=2,mode='wrap
')

denominator = scipy.ndimage.filters.convolve1d(nweight_2017, gaussian, axis=2,mode='wrap')

# avoid divide by 0 problems by setting zero values
# of the denominator to not a number (NaN)
denominator[denominator==0] = np.nan

interpolated_snow_17 = numerator/denominator


sweight = nweight_2017.sum(axis=2)
r,c = np.where(sweight == np.max(sweight))
plt.figure(figsize=(15,5))

#snow_2016[snow_2016 > 100] = np.nan

ipixel = 0 # To plot the i-th pixel
plt.plot((interpolated_snow_17)[r[ipixel],c[ipixel],:],'r--')
plt.plot((cover_2017)[r[ipixel],c[ipixel],:],'+')
plt.title('Interpolated Snowfall Sigma = 1.5: Year 2017')
plt.ylim(0,100)
```

Out[290]:

```
(0, 100)
```

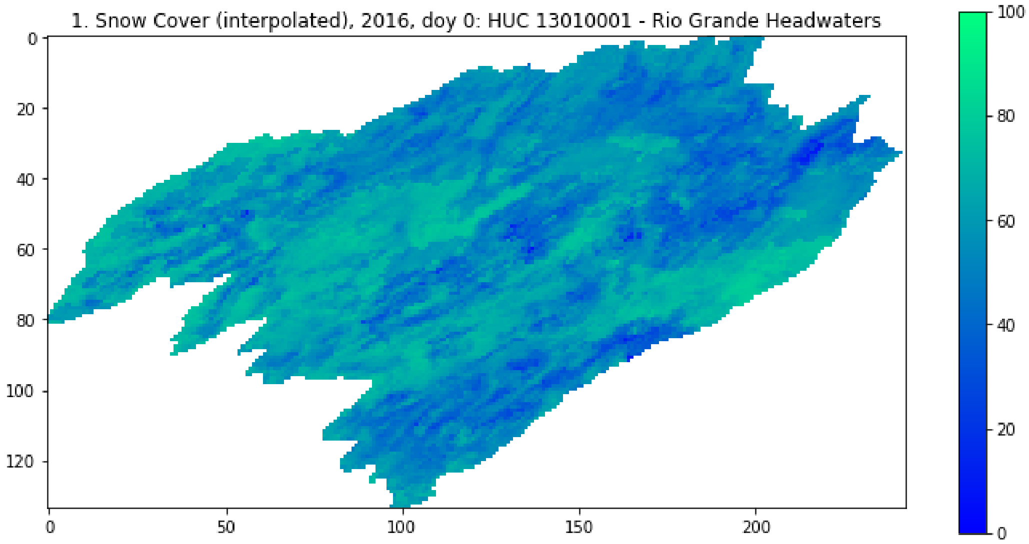


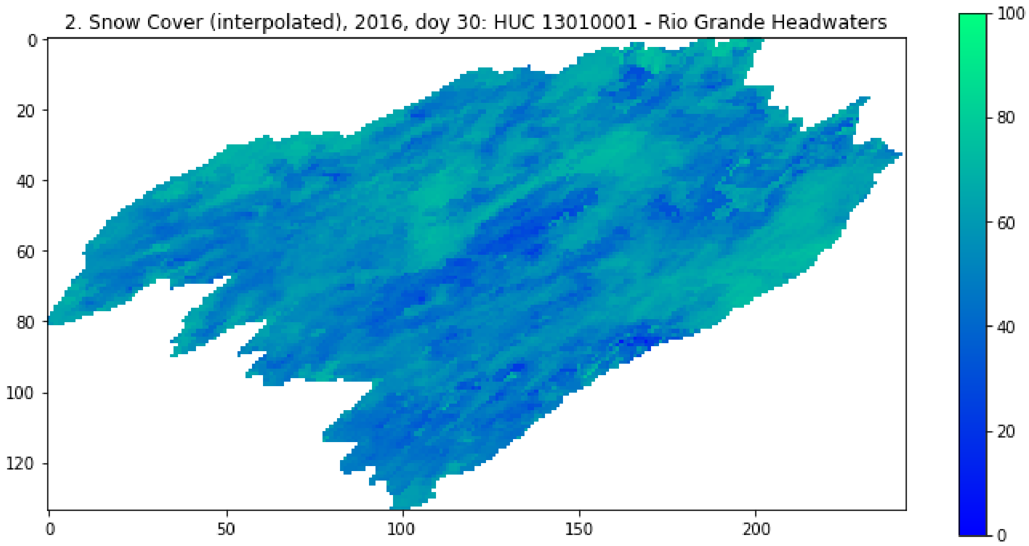
Interpolated Snow Cover 2016 - Graphical

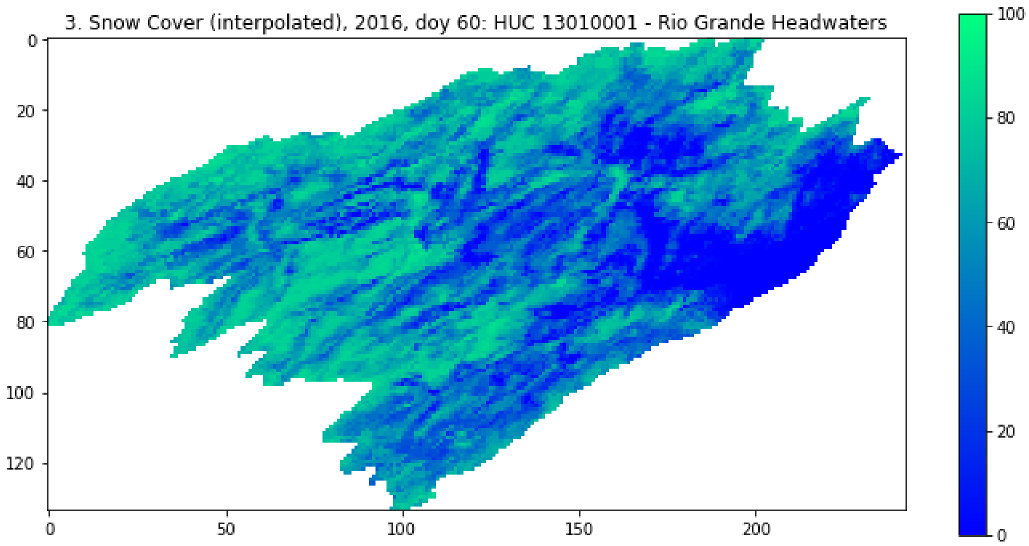
In [292]:

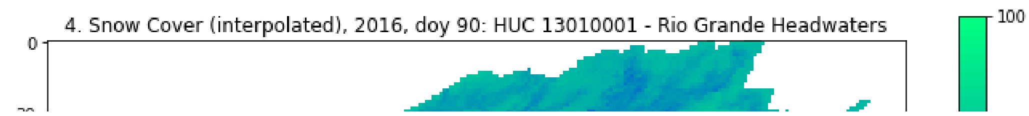
```
x = range(0, 366, 30)

for i in enumerate(x):
    plt.figure(figsize=(12, 12))
    plt.title(f'{i[0]+1}. Snow Cover (interpolated), 2016, doy {i[1]}: HUC 13010001 - Rio Grande H
```

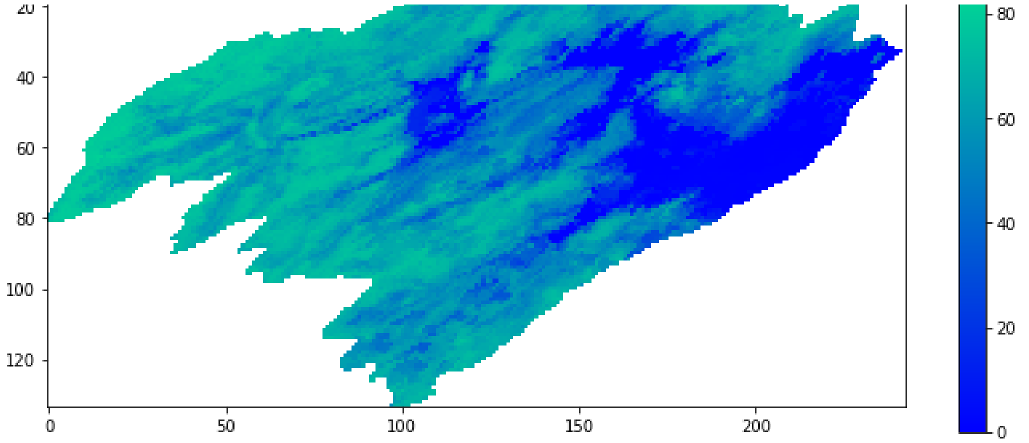
```
eadwaters')
    plt.imshow(interpolated_snow_16[...,i[1]], interpolation="nearest",vmin=0, vmax=100, cmap=plt.c
m.winter)

    plt.colorbar(shrink=0.5)
```



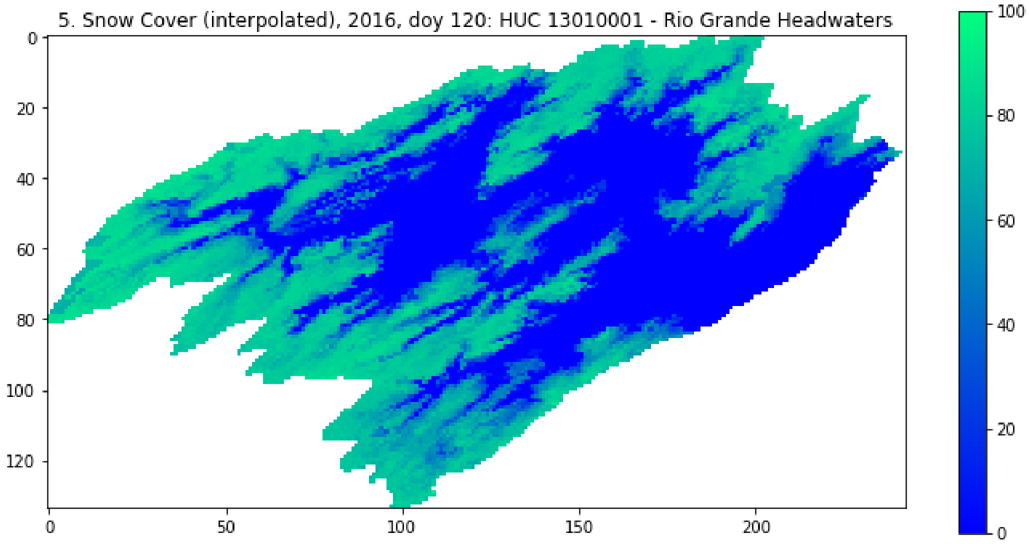




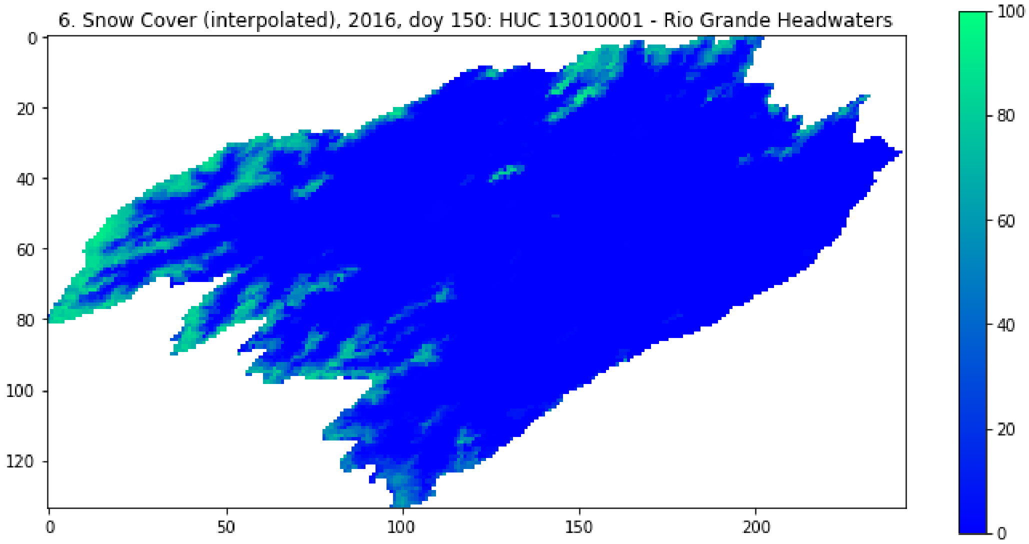




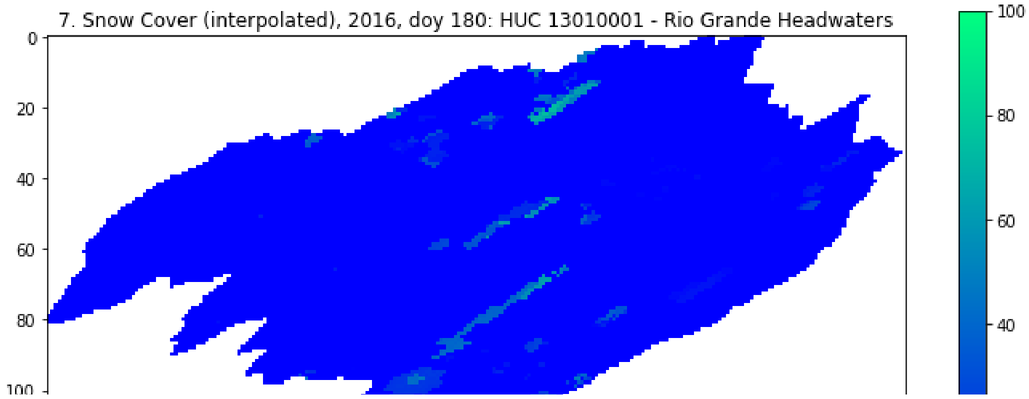
5. Snow Cover (interpolated), 2016, doy 120: HUC 13010001 - Rio Grande Headwaters

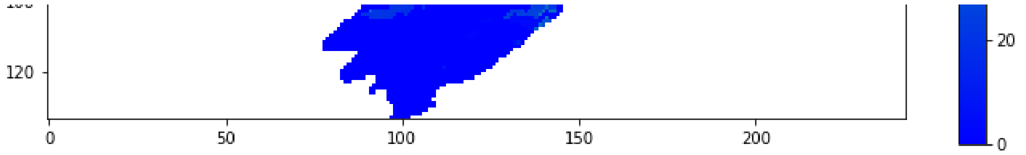


6. Snow Cover (interpolated), 2016, doy 150: HUC 13010001 - Rio Grande Headwaters

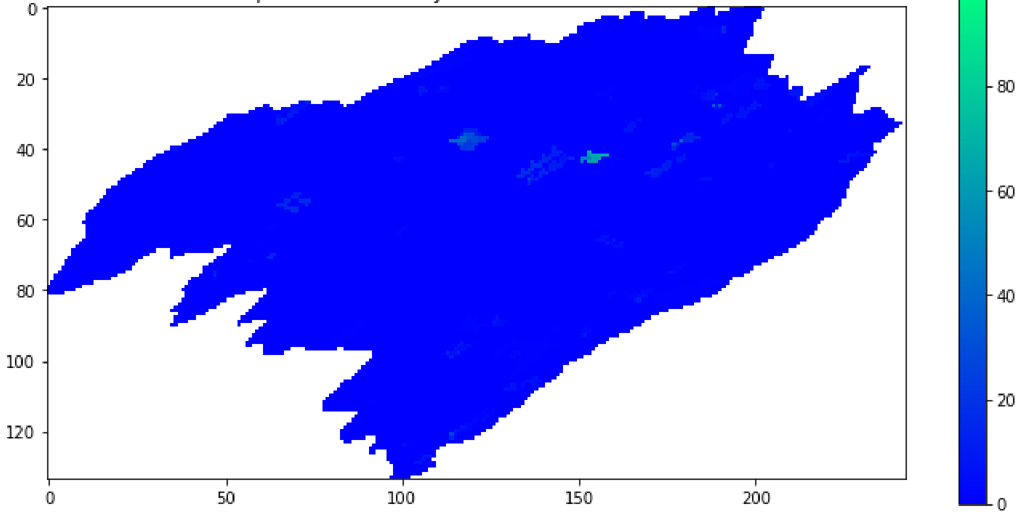


7. Snow Cover (interpolated), 2016, doy 180: HUC 13010001 - Rio Grande Headwaters

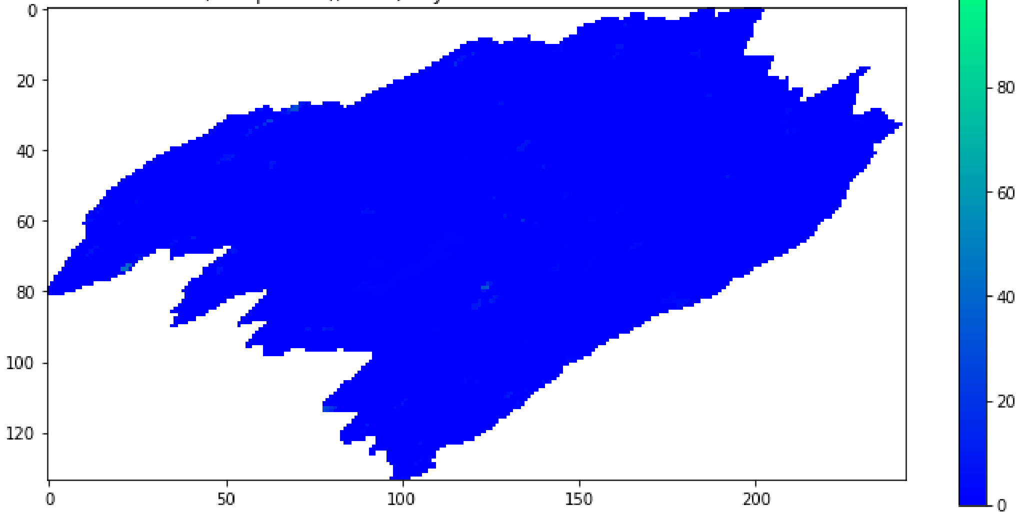




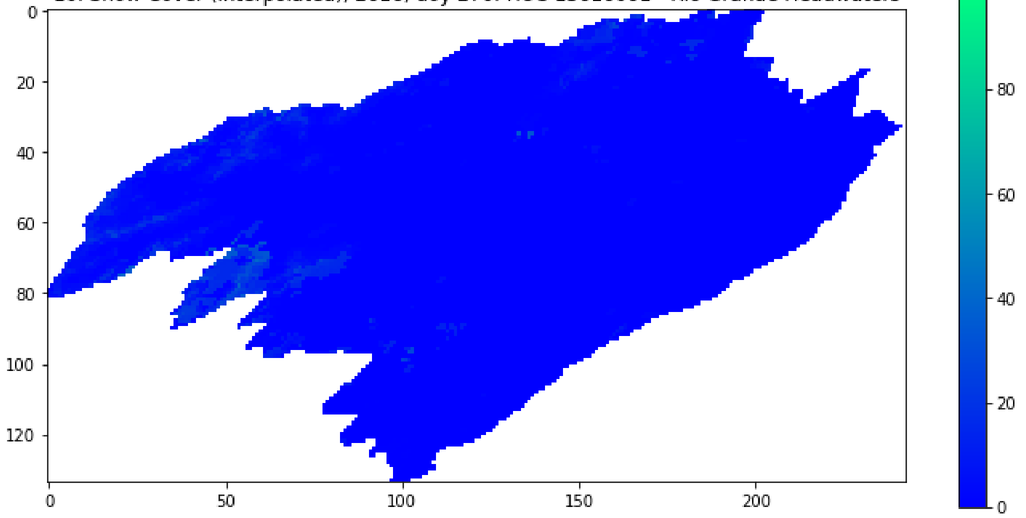
8. Snow Cover (interpolated), 2016, doy 210: HUC 13010001 - Rio Grande Headwaters



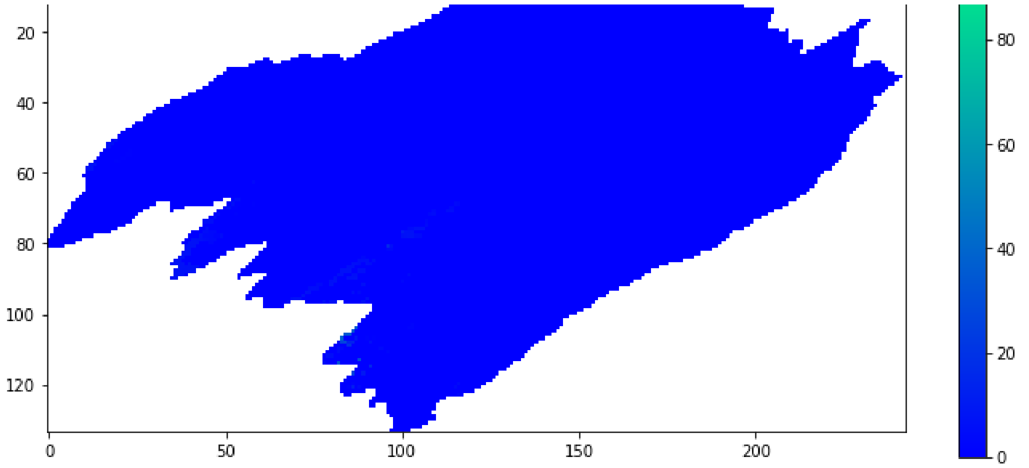
9. Snow Cover (interpolated), 2016, doy 240: HUC 13010001 - Rio Grande Headwaters

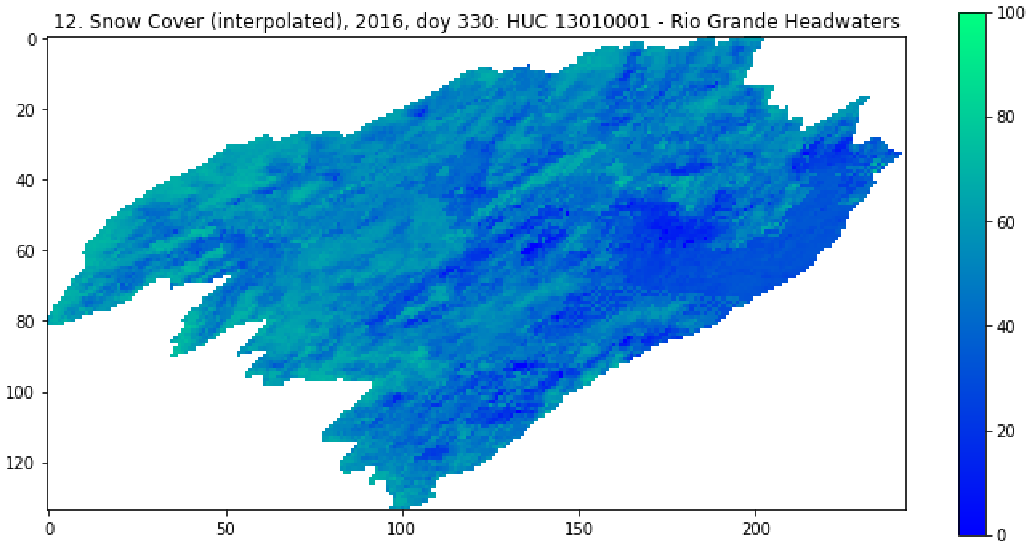


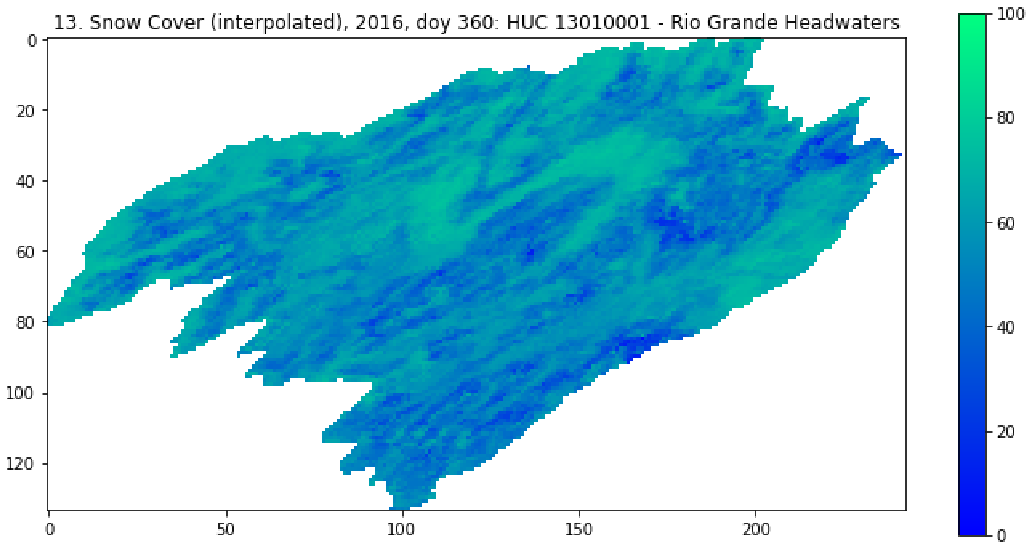
10. Snow Cover (interpolated), 2016, doy 270: HUC 13010001 - Rio Grande Headwaters



11. Snow Cover (interpolated), 2016, doy 300: HUC 13010001 - Rio Grande Headwaters





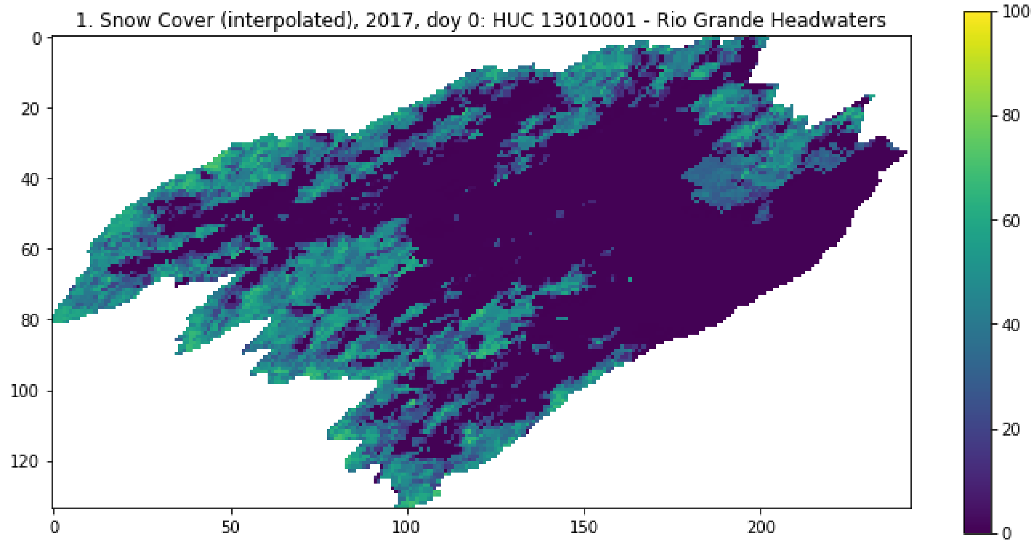


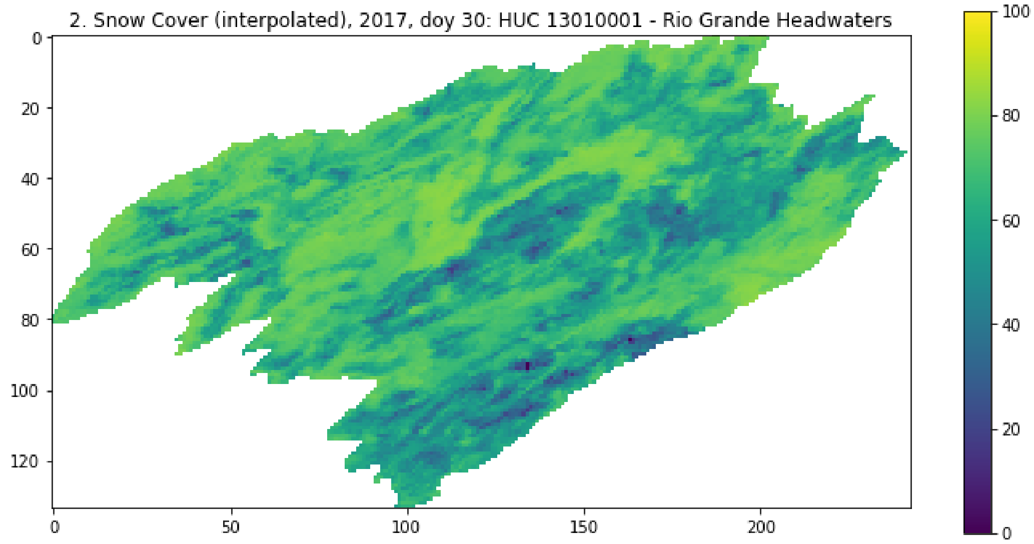
Interpolated Snow Cover 2017 - Graphical

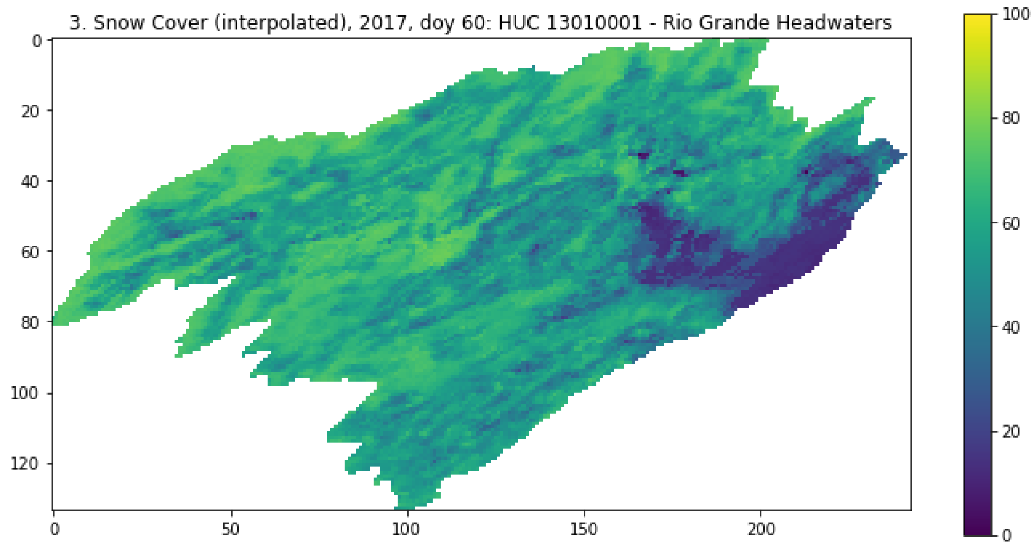
In [294]:

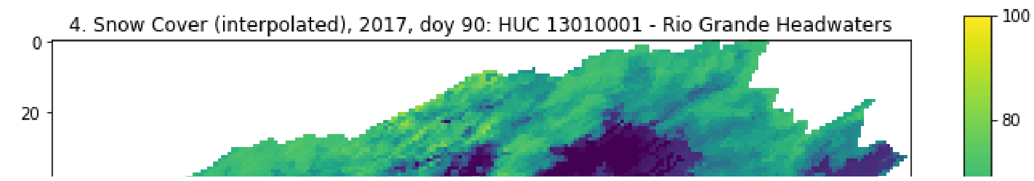
```
x = range(0, 365, 30)

for i in enumerate(x):
    plt.figure(figsize=(12, 12))
    plt.title(f'{i[0]+1}. Snow Cover (interpolated), 2017, doy {i[1]}: HUC 13010001 - Rio Grande H
eadwaters')
    plt.imshow(interpolated_snow_17[...,i[1]], interpolation="nearest",vmin=0, vmax=100, cmap=plt.c
m.viridis)
```

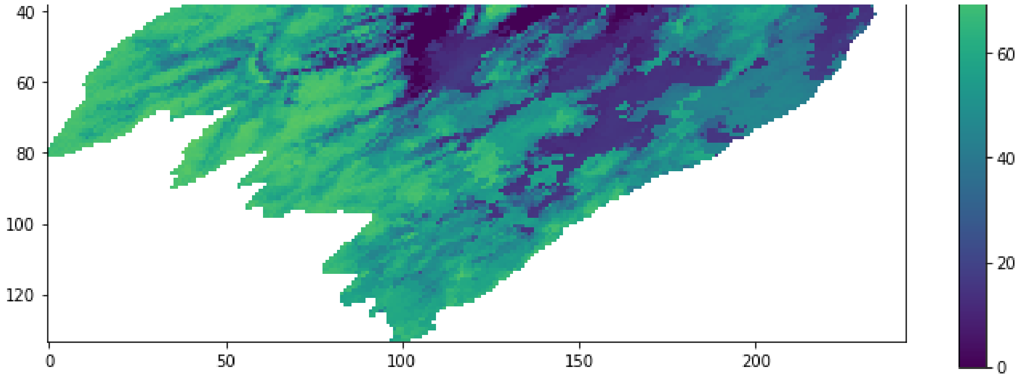
```
m.viridis)

    plt.colorbar(shrink=0.5)
```



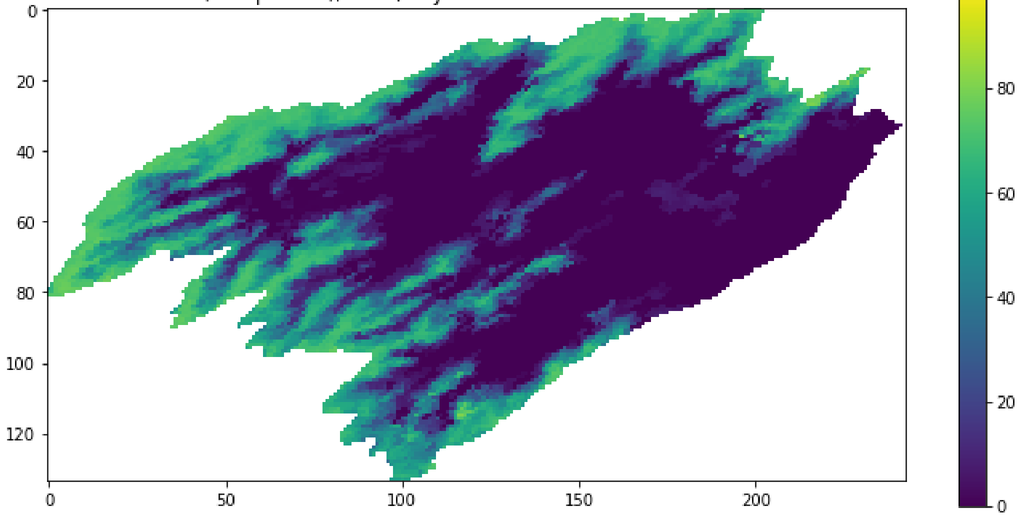




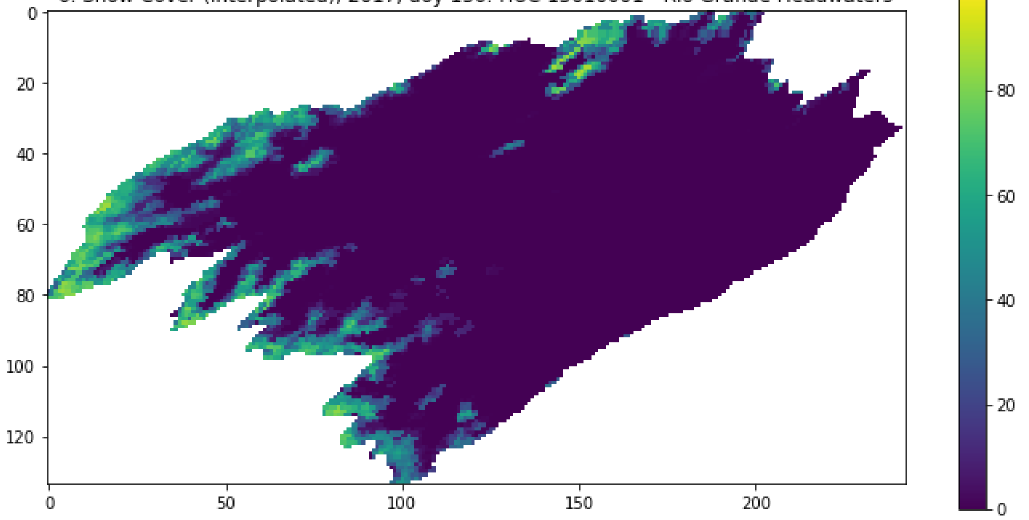




5. Snow Cover (interpolated), 2017, doy 120: HUC 13010001 - Rio Grande Headwaters

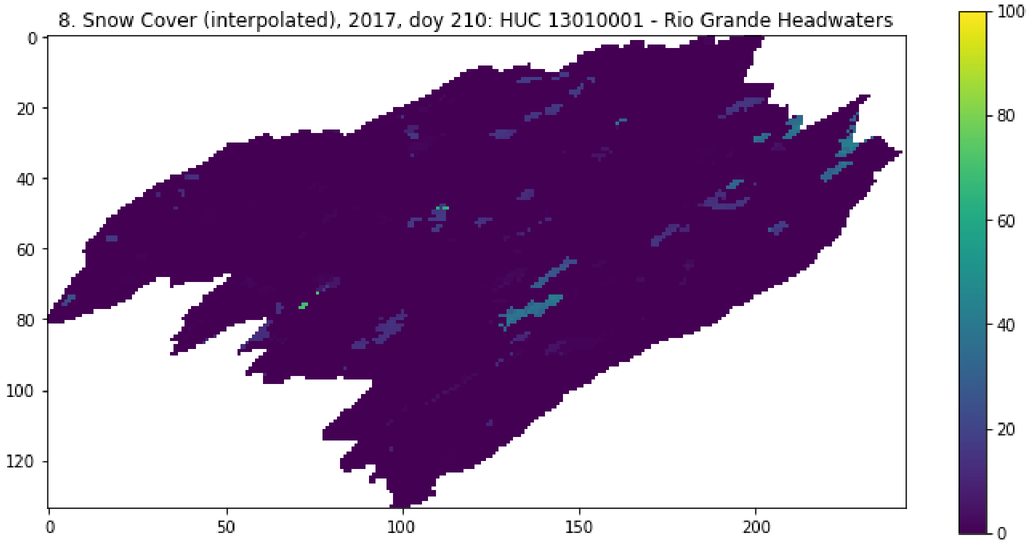


6. Snow Cover (interpolated), 2017, doy 150: HUC 13010001 - Rio Grande Headwaters

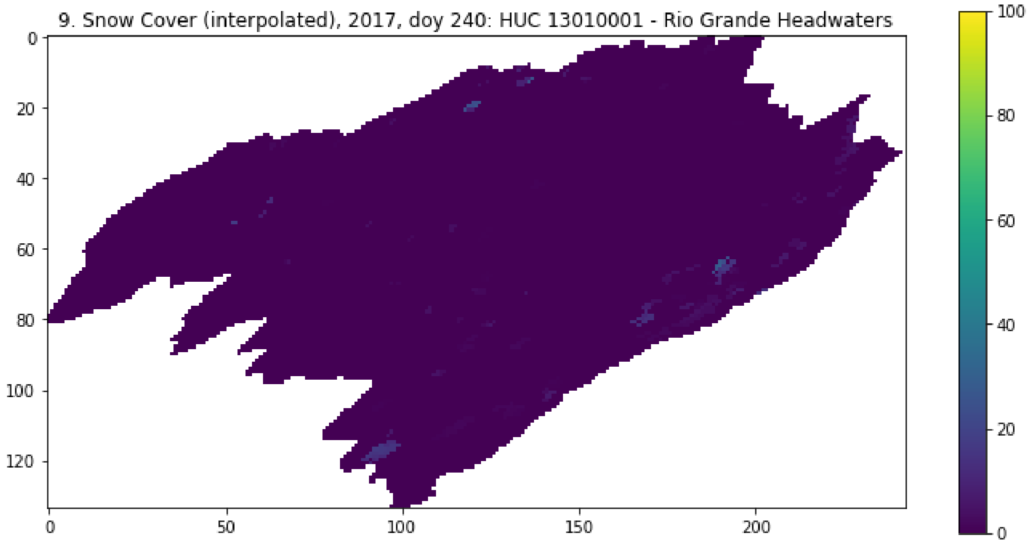


7. Snow Cover (interpolated), 2017, doy 180: HUC 13010001 - Rio Grande Headwaters

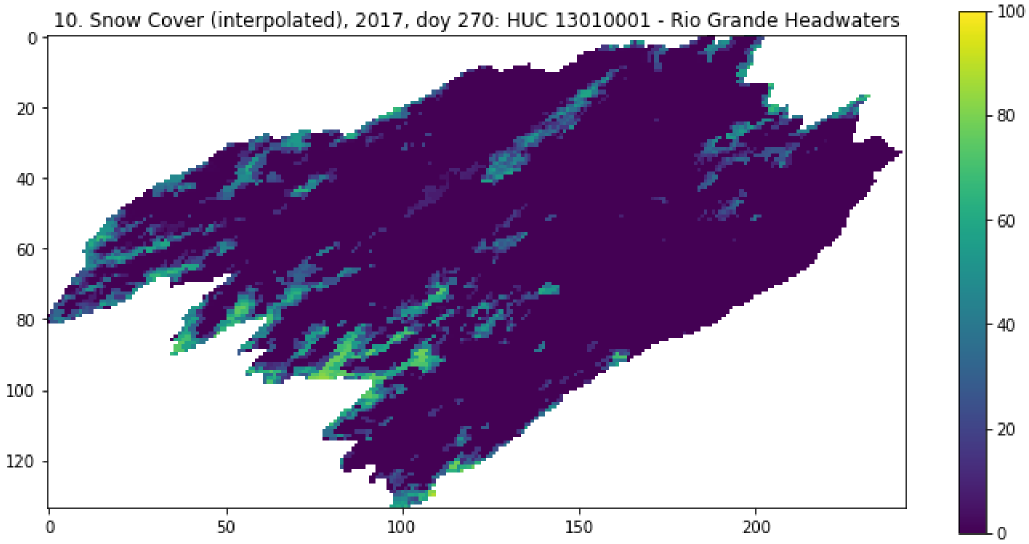
8. Snow Cover (interpolated), 2017, doy 210: HUC 13010001 - Rio Grande Headwaters



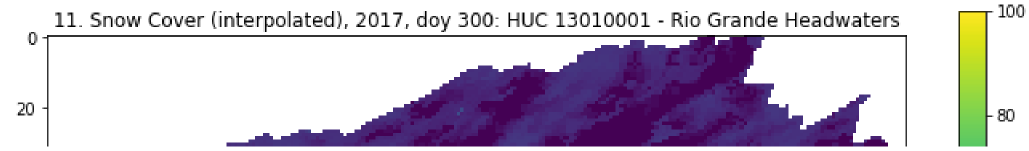
9. Snow Cover (interpolated), 2017, doy 240: HUC 13010001 - Rio Grande Headwaters

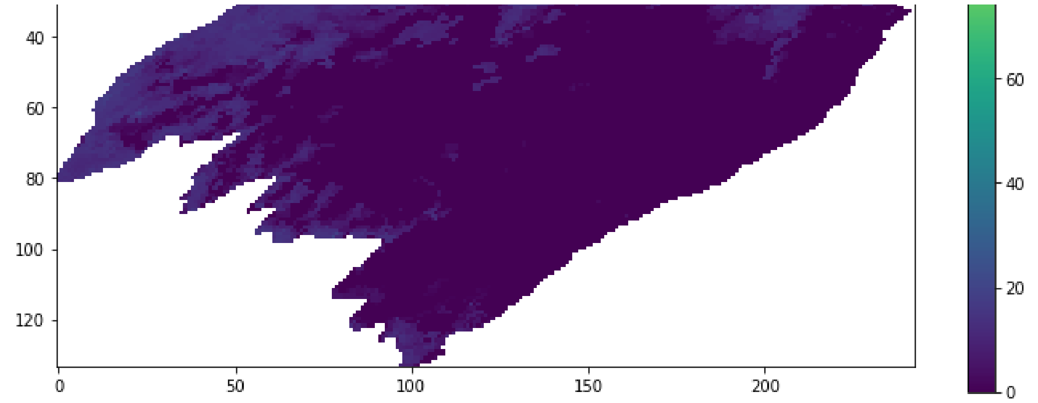


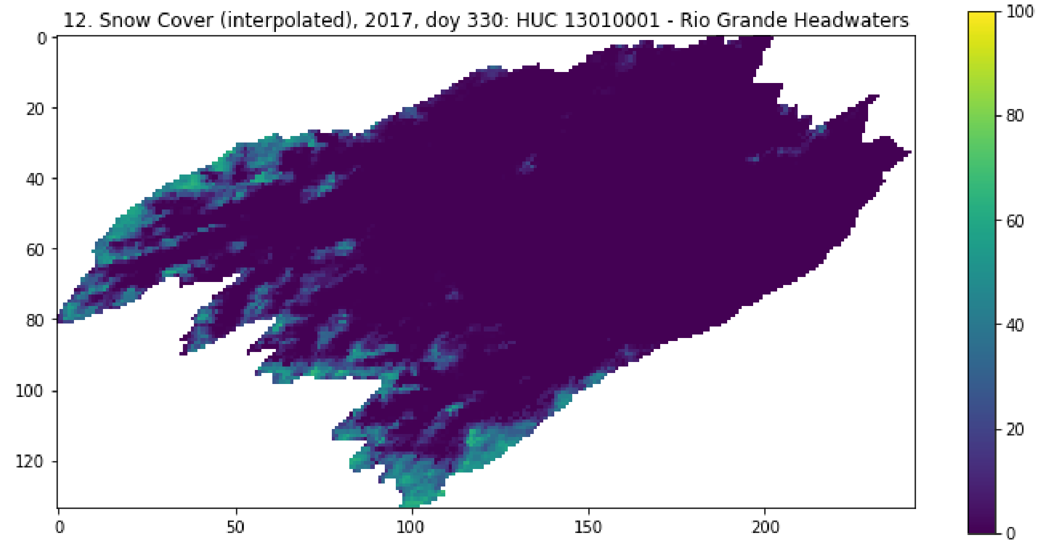
10. Snow Cover (interpolated), 2017, doy 270: HUC 13010001 - Rio Grande Headwaters

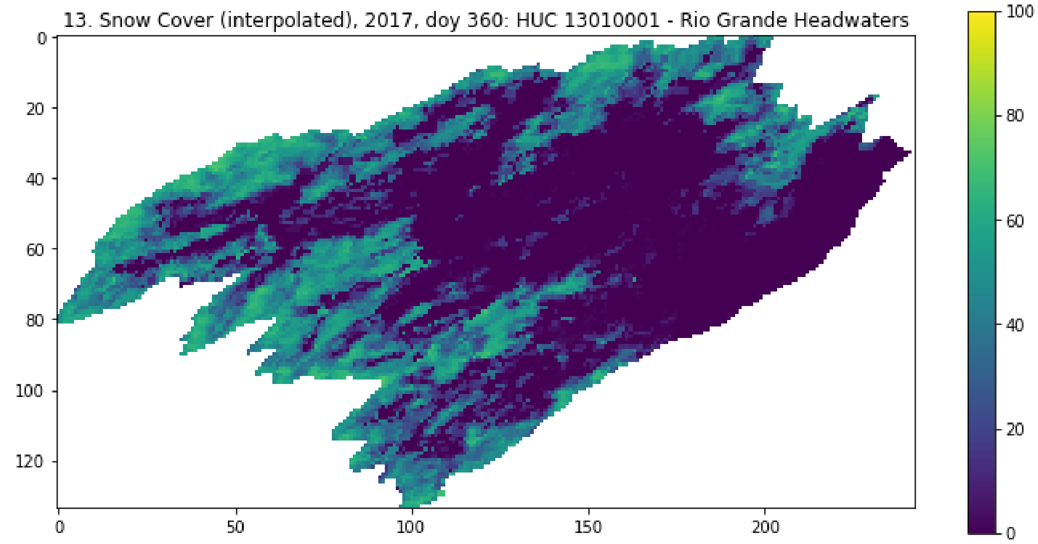


11. Snow Cover (interpolated), 2017, doy 300: HUC 13010001 - Rio Grande Headwaters









Analysis of Interpolation:

The interpolation with sigma value of 1.5 has definitely has a smoothing effect upon the data, some of the coarseness of the original MODIS resolution has been removed and for high contrast zones there appears to be a smoother gradient of change between areas of high and low snow cover.

Summary Statistics - Interpolated Snow Cover:

In [295]:

```
import pandas as pd
import numpy as np
```

```
pd.options.display.float_format = '{:,.2f}'.format

mean = []
mini = []
argmin = []
maxi = []
argmax = []
stdev = []

datasets = [interpolated_snow_16, interpolated_snow_17]


for data in datasets:
    data_mean = []
    for i in range(data.shape[2]):
        data_mean.append(np.nanmean(data[...,i]))
    data_mean = np.asarray(data_mean)
    mini.append(np.nanmin(data_mean))
    argmin.append(data_mean.argmin())
    maxi.append(np.nanmax(data_mean))
    argmax.append(data_mean.argmax())
    stdev.append(np.nanstd(data_mean))
    mean.append(np.nanmean(data_mean))



for dataset in datasets:

    mini.append(np.nanmin(dataset, axis=(0,1)))
    argmin.append(dataset.argmin())
    maxi.append(np.nanmax(dataset, axis=(0,1)))
    argmax.append(dataset.argmax())
    stdev.append(np.nanstd(dataset, axis=0, dtype=float))

df5 = pd.DataFrame({ "Year": range(2016,2018),
                    "Mean Snow Cover" : mean[0:2],
                    "StDev": stdev[0:2],
                    "Lowest Mean Snow Cover": mini[0:2],
                    "Low Time (doy)": argmin[0:2],
                    "Highest Mean Snow Cover": maxi[0:2],
                    "High Time (doy)": argmax[0:2]
                   })

df5
```

```
/opt/anaconda/envs/jupyterhub/lib/python3.6/site-packages/numpy/lib/nanfunctions.py:1545:
RuntimeWarning: Degrees of freedom <= 0 for slice.
  keepdims=keepdims)
```

Out[295]:

	Year	Mean Snow Cover	StDev	Lowest Mean Snow Cover	Low Time (doy)	Highest Mean Snow Cover	High Time (doy)
0	2016	25.08	23.85	0.00	189	63.13	11
1	2017	20.53	22.42	0.03	234	66.39	14

Loading Files & Presenting Basis For Hydrological Model:

In [296]:

```
# Collating the datasets and building a basic model overview of parameters.

filename16D = "discharge_DN_2016.npz"
filename16T = "DN2E_TPS_2016.npz"

filename17D = "discharge_DN_2017.npz"
filename17T = "DN2E_TPS_2017.npz"

#np.savez_compressed("discharge_DN_2016.npz", array1=array1, array2=array2, array3=array3)
```

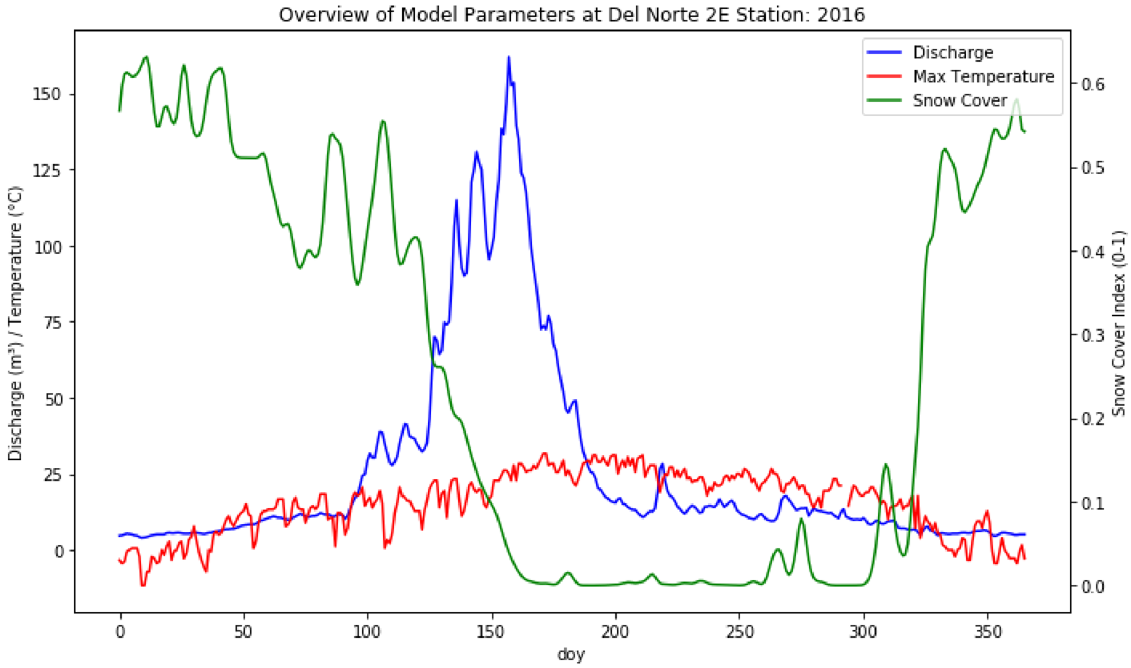
```
# Loading in the previously saved datasets and parsing to the correct variables:

discharge_data16 = np.load("discharge_DN_2016.npz")
data_dict = dict(discharge_data16)
discharge16 = data_dict["arr_0"]

discharge_data17 = np.load("discharge_DN_2017.npz")
data_dict = dict(discharge_data17)
discharge17 = data_dict["arr_0"]

temp_data16 = np.load("DN2E_TPS_2016.npz")
data_dict = dict(temp_data16)
temp16 = data_dict["arr_0"]
temp16 = temp16[1]
temp16 = (temp16 -32)*5/9

temp_data17 = np.load("DN2E_TPS_2017.npz")
data_dict = dict(temp_data17)
temp17 = data_dict["arr_0"]
temp17 = temp17[1]
temp17 = (temp17 -32)*5/9

# Creating a dataset of mean snow cover across the catchment for each day of each year:

mean_snow_16 = []
for i in range(interpolated_snow_16.shape[2]):
    mean_snow_16.append(np.nanmean(interpolated_snow_16[...,i]))

# Rescaling snow cover:
mean_snow_16 = np.asarray(mean_snow_16)
mean_snow_16 = mean_snow_16 * 0.01

mean_snow_17 = []
for i in range(interpolated_snow_17.shape[2]):
    mean_snow_17.append(np.nanmean(interpolated_snow_17[...,i]))

# Rescaling snow cover:
mean_snow_17 = np.asarray(mean_snow_17)
mean_snow_17 = mean_snow_17 * 0.01
```

In [297]:

```
header = ["discharge", "temperature", "snow_cover"]

arr_2016 = np.vstack((discharge16, temp16, mean_snow_16))
arr_2017 = np.vstack((discharge17, temp17, mean_snow_17))

data_16 = dict(zip(header, arr_2016))
data_17 = dict(zip(header, arr_2017))

filename16 = "Hydrological_Variables_2016"
filename17 = "Hydrological_Variables_2017"

# save the dataset
np.savez_compressed(filename16,**data_16)
np.savez_compressed(filename17,**data_17)
```

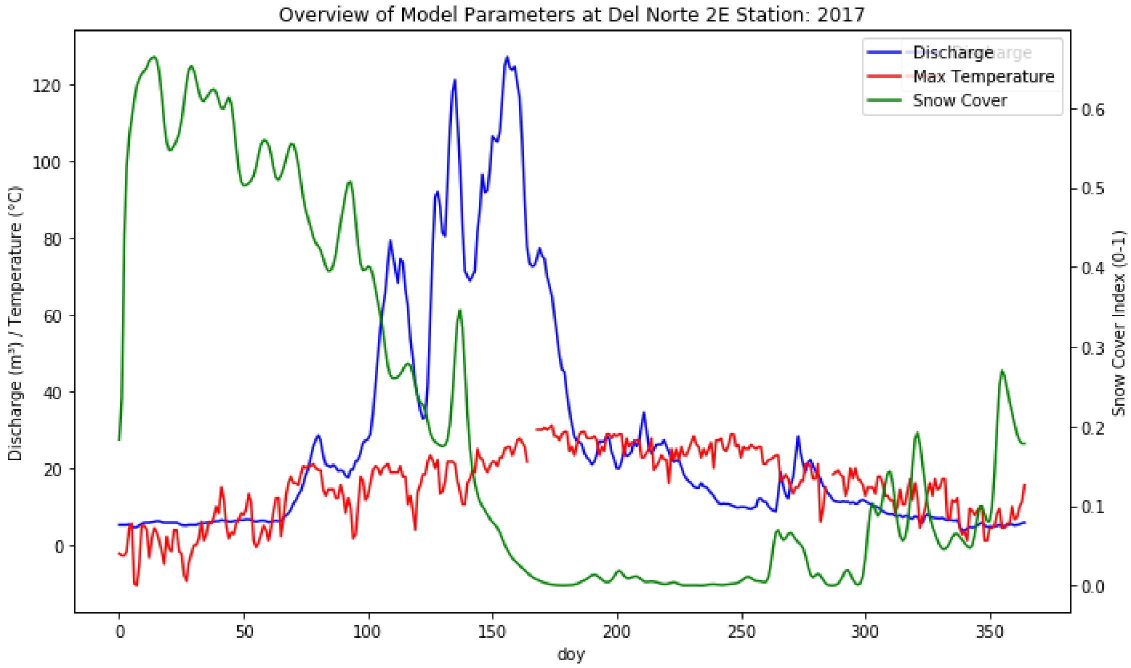
In [298]:

```
fig, ax1 = plt.subplots(figsize=(10,6))

ax1.set_xlabel('doy')
ax1.set_ylabel('Discharge (m³) / Temperature (°C)')
l1, =ax1.plot(discharge16, label="Discharge",color="b")
l2, =ax1.plot(temp16, label="Temperature", color="r")
ax2 = ax1.twinx()

ax2.set_ylabel('Snow Cover Index (0-1)')
l3, =ax2.plot(mean_snow_16, label="Snow Cover", color="g")

plt.legend([l1, l2, l3],["Discharge", "Max Temperature", "Snow Cover"], loc=1)
plt.title("Overview of Model Parameters at Del Norte 2E Station: 2016")
fig.tight_layout()
plt.show()
```



In [299]:

```
fig, ax1 = plt.subplots(figsize=(10,6))

ax1.set_xlabel('doy')
ax1.set_ylabel('Discharge (m³) / Temperature (°C)')
l1, =ax1.plot(discharge17, label="Discharge",color="b")
l2, =ax1.plot(temp17, label="Temperature", color="r")
plt.legend(loc="best")
ax2 = ax1.twinx()

ax2.set_ylabel('Snow Cover Index (0-1)')
l3, =ax2.plot(mean_snow_17, label="Snow Cover", color="g")

plt.legend([l1, l2, l3],["Discharge", "Max Temperature", "Snow Cover"], loc=1)
plt.title("Overview of Model Parameters at Del Norte 2E Station: 2017")
fig.tight_layout()
plt.show()
```



Analysis:

Analysis:

The graphs above provide an excellent visual representation of the hydrological flux within the HUC 1301001 catchment throughout each year. The flux in snow cover is clearly sensitive to the seasonal changes in temperature with melt events being recorded first in temperature, through snow cover, as discharge pulses.

The data collected should provide a good basis for a hydrological model based upon the system if a visual guide of the relationships between the variables is trustworthy.