GDAL_in_Python

November 8, 2022

In today's exercise we will learn how to use Python to extract data from numerous files using different Python packages. The aim is to calculate normalized difference vegetation index (NDVI) and compare it to two climate variables: air temperature and precipitation.

NDVI can be used as a proxy for healthy living vegetation. In our case NDVI ranges between 0 - 1, where 0 means non-vegetation surface and 1 means living vegetation.

I think most of you remember late spring and summer of 2018 as a particularly dry time period. Therefore, we will use Landsat (satellite) images from this time period to analyze the NDVI together with air temperature and precipitation.

The Landsat images has been retrieved from https://earthexplorer.usgs.gov/

Air temperature has been retrieved from https://psl.noaa.gov/data/gridded/data.ghcncams.html
Variable: air temperature (Kelvin)
Spatial resolution: 0.5 degrees
Temporal resolution: Monthly mean

Precipitation has been retrieved from https://psl.noaa.gov/data/gridded/data.gpcc.html
Variable: Monthly precipitation
Spatial resolution: 1.0 degrees
Temporal resolution: Monthly total

For Windows users:

Open OSGeo4W Shell (Start menu –> OSGeo4W –> OSGeo4W Shell).

Write:

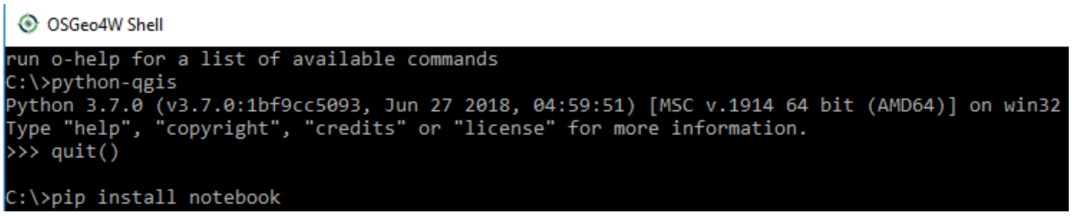
o4w_env

This will set necessary PATHs for OSGeo4W Shell to know where Python and QGIS Python packages are located.

Jupyter Notebook should already be installed on GIS lab computers. If you are on your private computer and need to install Jupyter Notebook, follow the steps below:

pip install notebook

in terminal:



We will also analyze NetCDF files with the xarray module which has a dependency in the NetCDF4 module, which means that we need to install NetCDF4 in the same way as with notebook:

pip install NetCDF4

and after this, also xarray:

pip install xarray

We need to install these packages on the GIS lab computers as well.

Now, when we have set all PATHs for Python and necessary QGIS packages, and installed Jupyter Notebook and/or NetCDF4 and xarray, we want to locate ourselves in a directory where we can also save or Jupyter Notebook project. Therefore we have to move around a bit in the terminal. By default we are likely located under C:. To move around we have to use some commands in the shell prompt: cd = move to other path or directory cd.. = go back one step in path mkdir = create folder

If we are currently in C:\ we can create a folder under C:\ called exercise3 by writing:

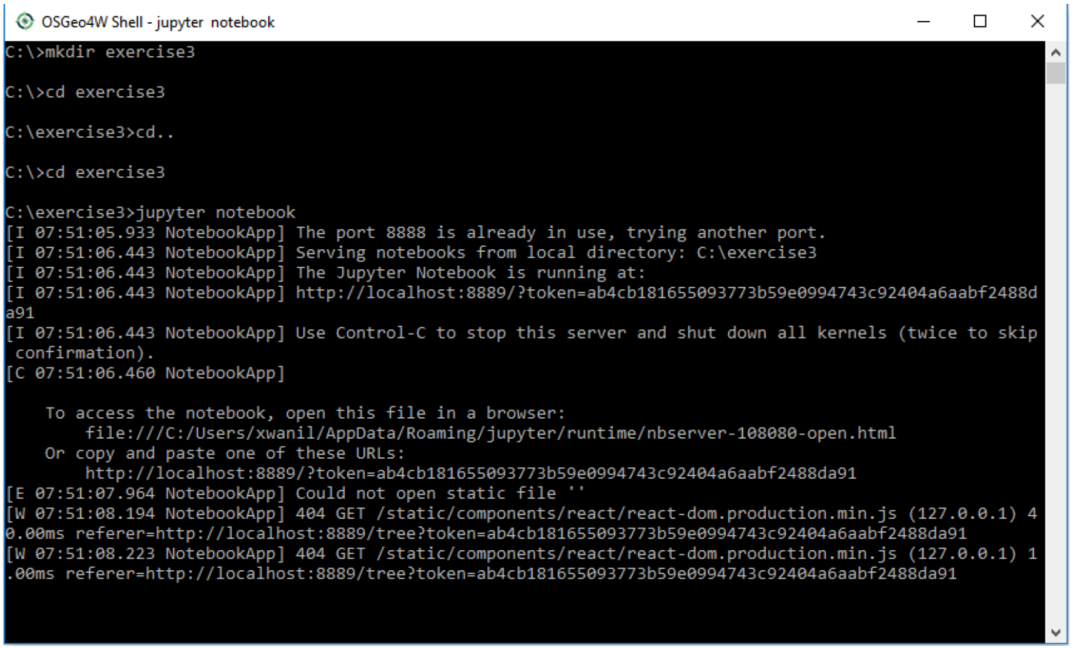
mkdir exercise6

We can then move into this folder by writing:

cd exercise3

From here we can start our Jupyter Notebook project by writing:

jupyter notebook



For Mac users:

Instructions on Canvas

Exercise:

When we created our directory where we want to save our Jupyter Notebook project and Jupyter Notebook is up and running, we can start with the exercise.

In this exericse we will use the following Python packages:

[1]:
```
# Import packages
# The sys is a module with variables and functions that work closely with the␣
 ↪interpreter
# https://docs.python.org/3/library/sys.html
import sys

# The os module is useful if you want to create folders, find paths, list files␣
 ↪in paths, etc.
# https://docs.python.org/3/library/os.html
import os

# glob is similar to os but with Unix style pathnames, etc.
# https://docs.python.org/3/library/glob.html
import glob

# The tarfile module is used to extract tar files. Tar files are similar to␣
 ↪zip, i.e. compressed file archives
# https://docs.python.org/3/library/tarfile.html
```

```
import tarfile

# numpy is one of the most important modules for scientific programming in␣
 ↪Python
# https://numpy.org/
import numpy as np

# xarray is a nice module for handling NetCDF files. Has NetCDF4 as a␣
 ↪dependency, which means it also needs the
# NetCDF4 module to work properly.
# http://xarray.pydata.org/en/stable/
import xarray as xr

# matplotlib is a fundamental module for plotting graphs, maps, etc.
# https://matplotlib.org/
import matplotlib.pyplot as plt

# osgeo contains e.g. gdal and ogr
from osgeo import gdal, ogr

# osgeo.gdalconst contains some constants for gdal
from osgeo.gdalconst import *
```

[2]:
```
# Defining some directories and paths

# Directory with compressed Landsat images
directory = 'C:/Exercise6/LandsatGoteborg/'

# Check if directory with compressed files exists!
if not os.path.exists(directory):
    print('Error! Directory does not exist! Check path!')
else:
    print('Yippie! Exists!')

# Output directory for extracted files
output = 'C:/Exercise6/Output/'

if not os.path.exists(output):
    os.makedirs(output)

# Create temp folder for temp files
output_temp = 'C:/Exercise6/temp/'

if not os.path.exists(output_temp):
    os.makedirs(output_temp)
```

```
Yippie! Exists!
```

[3]:
```
# Make a list with all files in directory
flist = glob.glob(directory + '*.tar.gz')

# First we need to unpack all files into our output path
for fname in flist:
    tar = tarfile.open(fname, 'r:gz')
    tar.extractall(path=output)
    tar.close()

# Same as above, but more Pythonic.
for fname in flist:
    with tarfile.open(fname, 'r:gz') as tar:
        tar.extractall(path=output)
```

[4]:
```
# The extracted files are quite big.
# We want to look at Gothenburg, so we want to cut them a bit.
# Again, we need to get a list of filenames, but now for extracted files
fnames = glob.glob(output + '*B4.TIF')

# We also need coordinates for a bounding box around Gothenburg with which we␣
 ↪can cut
lonmin = 300000 # Five zeros
latmin = 6385000 # Three zeros
lonmax = 340000 # Four zeros
latmax = 6430000 # Four zeros

# Create an array for bounding box
bbox = (lonmin, latmax, lonmax, latmin)

# Again, a new folder, now for cropped images
output_crop = 'C:/Exercise6/crop/'

# Create if it does not exist
if not os.path.exists(output_crop):
    os.makedirs(output_crop)

# A loop to cut files with GDAL
for fname in fnames:
    extension = os.path.splitext(fname)
    temp_filename = fname.split('B4.TIF')[0]
    temp_filename_out = temp_filename.split('Output')[1]
    ds1 = gdal.Translate(output_crop + temp_filename_out + 'B4_SA.TIF',␣
 ↪temp_filename + 'B4.TIF', projWin=bbox)
    ds2 = gdal.Translate(output_crop + temp_filename_out + 'B5_SA.TIF',␣
 ↪temp_filename + 'B5.TIF', projWin=bbox)
```

Next we calculate NDVI...

NDVI stands for Normalized Difference Vegetation Index.

To calculate NDVI we need near infrared (NIR) and visible red visible (VIS) images. NDVI can then be calculated as follows:

NDVI = (NIR - VIS)/(NIR + VIS)

In our data:

NIR = files ending with B5 VIS = files ending with B4

This gives us:

NDVI = (B5 - B4)/(B5 + B4)

[5]:
```
# Now we need a folder for our NDVI files....

output_ndvi = 'C:/Exercise6/ndvi/'

# Create output folder if it does not exist...
if not os.path.exists(output_ndvi):
    os.makedirs(output_ndvi)

# For loop to calculate NDVI based on the filesnames in fnames (from previous␣
 ↪cell). We add B4_SA.TIF and B5_SA.TIF,
# which are the files endings of our cropped files of the Gothenburg area
for fname in fnames:
    temp_filename = fname.split('B4.TIF')[0]
    temp_filename_crop = temp_filename.split('Output')[1] # Careful, maybe you␣
 ↪have a small o in our 'Output'

    # Load raster data. You remember this from yesterday
    dataSet = gdal.Open(output_crop + temp_filename_crop + 'B4_SA.TIF')
    VIS = dataSet.ReadAsArray().astype(float)
    dataSet = gdal.Open(output_crop + temp_filename_crop + 'B5_SA.TIF')
    NIR = dataSet.ReadAsArray().astype(float)

    # NIR = B5
    # VIS = B4

    # Calculate NDVI
    ndvi = (NIR - VIS)/(NIR + VIS)

    # Create raster data
    rows = dataSet.RasterYSize
    cols = dataSet.RasterXSize

    outDs = gdal.GetDriverByName('GTiff').Create(output_ndvi +␣
 ↪temp_filename_crop + 'NDVI.TIF', cols, rows, int(1), GDT_Float32)
    outBand = outDs.GetRasterBand(1)
```

```
    # Write ndvi to raster
    outBand.WriteArray(ndvi, 0, 0)

    # Set georeference and projection to same as previous files (NIR and VIS)
    outDs.SetGeoTransform(dataSet.GetGeoTransform())
    outDs.SetProjection(dataSet.GetProjection())

    del dataSet, outDs, outBand

#dataSet = None
#del dataSet1, dataSet2
```

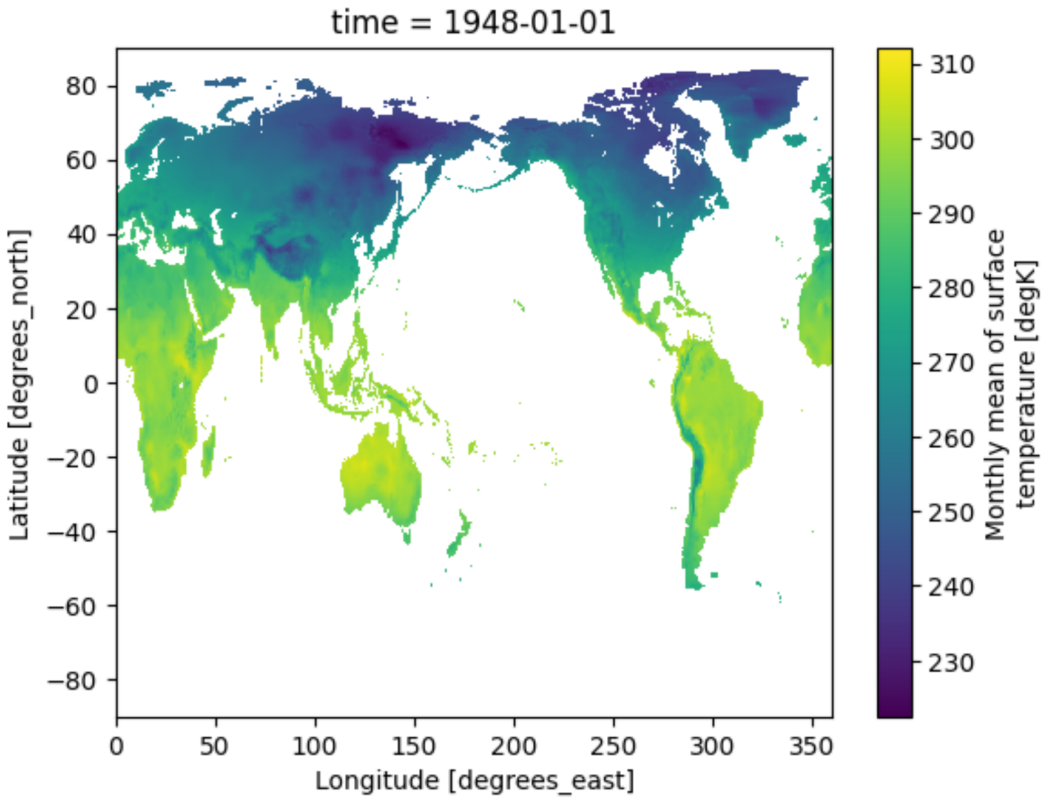
Now we want to do a comparison of NDVI between an urban and a rural area. For this we need two polygons; one for an urban area and one for a rural area. Create these in QGIS. You can do this by adding e.g. a basemap, to differentiate between urban and rural areas. It is also possible to add one of the cropped images (ndvi images). When you have your polygon layers we can continue with the coding:

[6]:
```
# Make comparison between urban and rural areas
ndvifiles = glob.glob(output_ndvi +'/*.tif')

# Create array to store values which we will later save to textfile.
# We will save date from filename, mean for urban and mean for rural, which␣
 ↪defines number of columns.
# One value for each file which then defines our number of rows.
result = np.zeros([ndvifiles.__len__(), 3]) # two underscores len two␣
 ↪underscores ()

# Path to shapefiles directory
shapeDir = 'C:/Exercise6/Shapefiles/'

# Path to actual shapefiles
urbanShape = shapeDir + 'urban.shp'
ruralShape = shapeDir + 'rural.shp'

# a counter for our loop
idx = 0

# For loop to estimate mean NDVI for urban and rural areas
for fname in ndvifiles:
    result[idx, 0] = int(fname[-32:-26])
    #print(fname[-32:-26])

    # Urban area
    gdal.Warp(output_temp + 'cut_ndvi_u.tif', fname, cutlineDSName=urbanShape,␣
 ↪cropToCutline=True, dstNodata=-9999)
```

```
    dataSet = gdal.Open(output_temp + 'cut_ndvi_u.tif')
    nd = dataSet.GetRasterBand(1).GetNoDataValue()
    mat = np.array(dataSet.ReadAsArray())
    bol = mat != nd
    result[idx, 1] = np.mean(mat[bol])

    # Rural area
    gdal.Warp(output_temp + 'cut_ndvi_r.tif', fname, cutlineDSName=ruralShape,␣
 ↪cropToCutline=True, dstNodata=-9999)
    dataSet = gdal.Open(output_temp + 'cut_ndvi_r.tif')
    nd = dataSet.GetRasterBand(1).GetNoDataValue()
    mat = np.array(dataSet.ReadAsArray())
    bol = mat != nd
    result[idx, 2] = np.mean(mat[bol]) # Mean for rural area, but how? We need␣
 ↪more code above
    idx += 1

# Table header
result_header = 'time urban rural'

# How your numbers should be presented (integers, decimals, etc)
result_fmt = '%1d', '%6.3f', '%6.3f'

# Path and filename where we want to save our txt
result_path = 'C:/Exercise6/result.txt'

np.savetxt(result_path, result, delimiter=', ', fmt=result_fmt,␣
 ↪header=result_header)
```

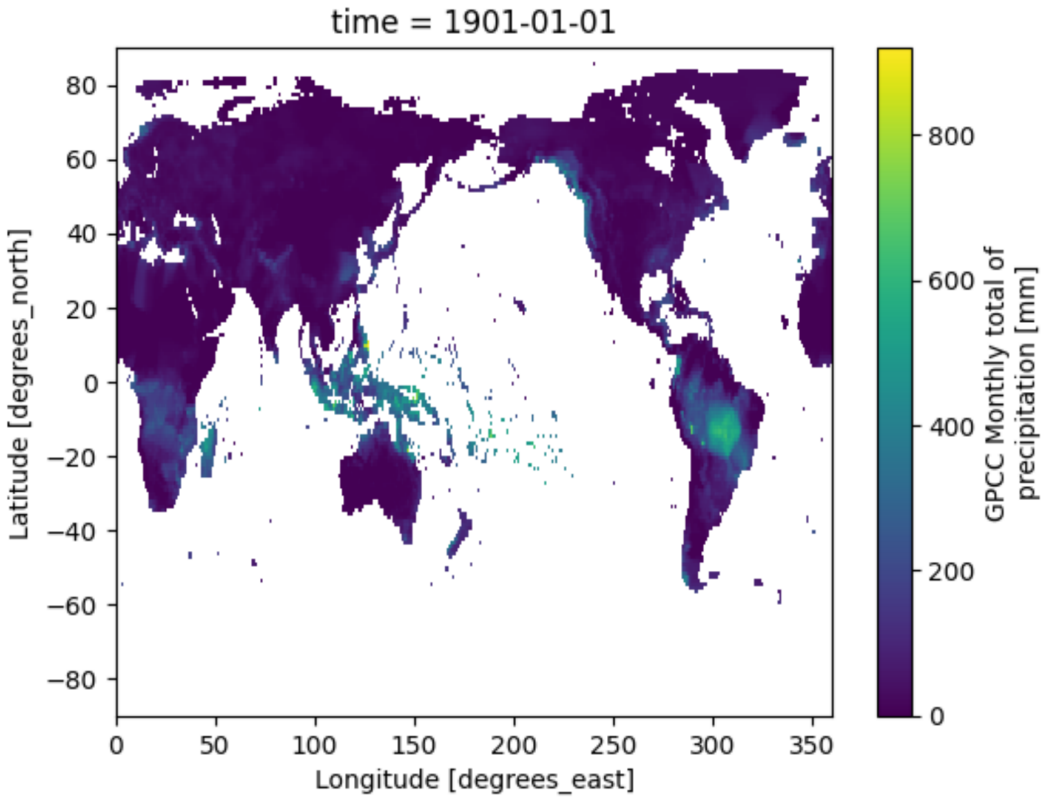
[7]:
```
# Now we also want to look at some climate data from our studied time period
# The data is stored in netCDF files (*.nc)

# Path to file with air temperature
airnc = 'C:/Exercise6/climate_data/air.mon.mean.nc'

# Path to file with precipitation
precnc = 'C:/Exercise6/climate_data/precip_mon_mean.nc'

# Load the netCDF file with air temperature data using xarray
airtemperature = xr.open_dataset(airnc)

# Try to plot the data. What do you see? What date is this time step for?
# What happens if you change time=0 to something else, e.g. time=845
air = airtemperature.air.isel(time=0)
air.plot()
plt.show()
```

```
# Get longitude and latitude data from the netCDF
lon = airtemperature['lon']
lat = airtemperature['lat']

# Coordinates for Gothenburg
latgbg = 57.71
longbg = 11.96

# Find grid closest to coordinates for Gothenburg
lon_idx = (np.abs(lon - longbg)).argmin()
lat_idx = (np.abs(lat - latgbg)).argmin()

# Get timeseries data for grid representing Gothenburg
Tair_gbg = airtemperature.sel(lon=lon[lon_idx]).sel(lat=lat[lat_idx])

# Get data for 2017 and 2018
Tair_gbg_2017_2018 = Tair_gbg.sel(time=Tair_gbg.time.dt.year.isin([2017,2018]))

# Get 30 year data (1981-2010)
Tair_gbg_1981_2010 = Tair_gbg.sel(time=slice("1981-01-01","2010-12-31"))

# Create monthly mean values for 2017-2018 and 1981-2010
Tair_gbg_2017_2018_mon_mean = Tair_gbg_2017_2018.groupby(Tair_gbg_2017_2018.
 ↪time.dt.month).mean("time").air.values
Tair_gbg_1981_2010_mon_mean = Tair_gbg_1981_2010.groupby(Tair_gbg_1981_2010.
 ↪time.dt.month).mean("time").air.values
```



[8]:
```
# Load the netCDF file with precipitation data using xarray
precipitation = xr.open_dataset(precnc)

# Plot data
prec = precipitation.precip.isel(time=0)
prec.plot()
plt.show()

# Latitude and longitude data from netCDF for precipitation
lon = precipitation['lon']
lat = precipitation['lat']

# Find closest grid for Gothenburg
lon_idx = (np.abs(lon - longbg)).argmin()
lat_idx = (np.abs(lat - latgbg)).argmin()

# Time series data
Prec_gbg = precipitation.sel(lon=lon[lon_idx]).sel(lat=lat[lat_idx])

# Data for 2017-2018 and 1981-2010
```

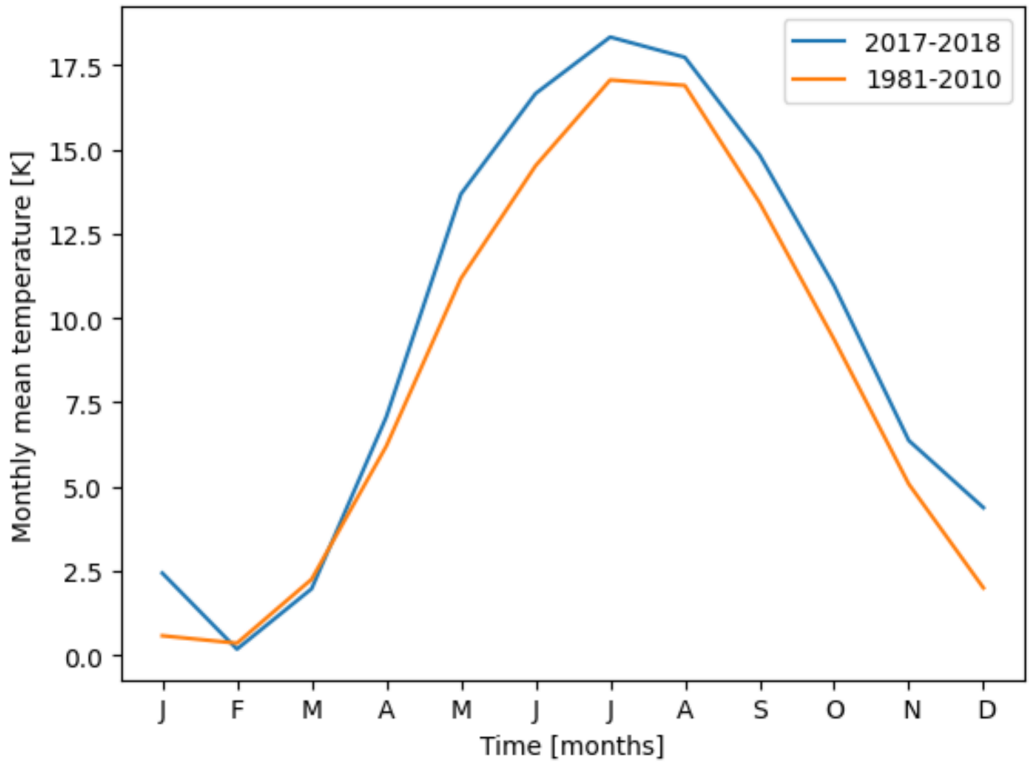
```
Prec_gbg_2017_2018 = Prec_gbg.sel(time=Prec_gbg.time.dt.year.isin([2017,2018]))
Prec_gbg_1981_2010 = Prec_gbg.sel(time=slice("1981-01-01","2010-12-31"))

# Monthly mean values
Prec_gbg_2017_2018_mon_mean = Prec_gbg_2017_2018.groupby(Prec_gbg_2017_2018.
  ↪time.dt.month).mean("time").precip.values
Prec_gbg_1981_2010_mon_mean = Prec_gbg_1981_2010.groupby(Prec_gbg_1981_2010.
  ↪time.dt.month).mean("time").precip.values
```

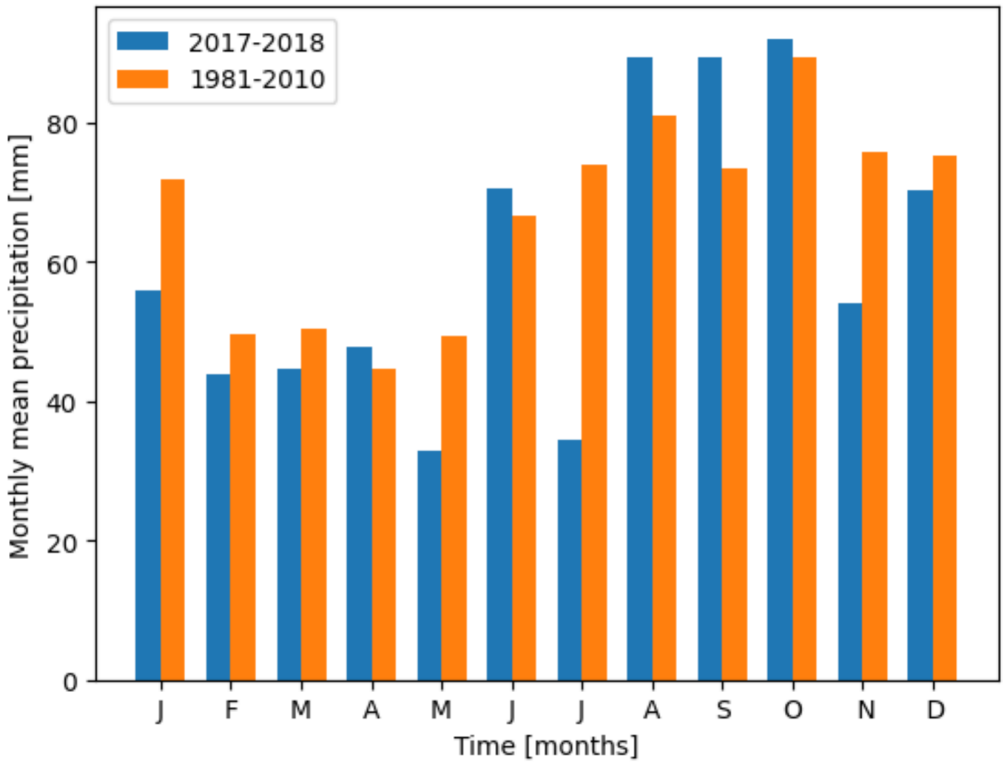


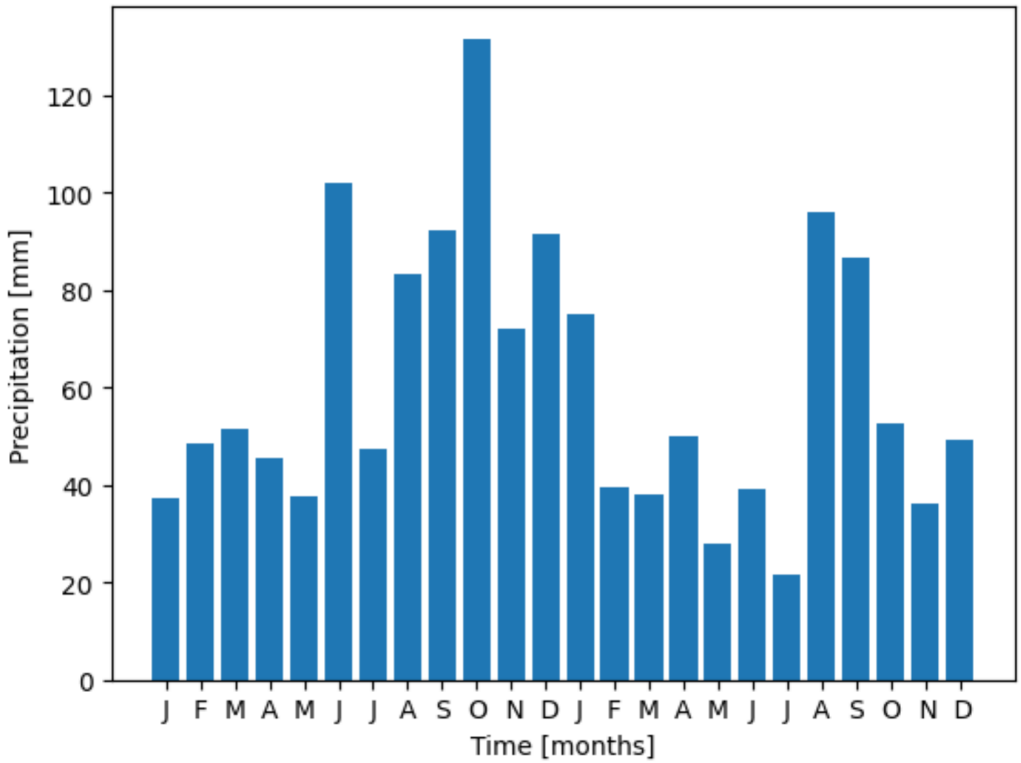
[9]:
```
# Now we want to plot the data above
# Plot air temperature timeseries
x = np.arange(Tair_gbg_1981_2010_mon_mean.__len__())
plt.figure(1)
plt.plot(x, Tair_gbg_2017_2018_mon_mean-273.15, label='2017-2018')
plt.plot(x, Tair_gbg_1981_2010_mon_mean-273.15, label='1981-2010')

# Labels
plt.xlabel('Time [months]')
plt.ylabel('Monthly mean temperature [K]')
```

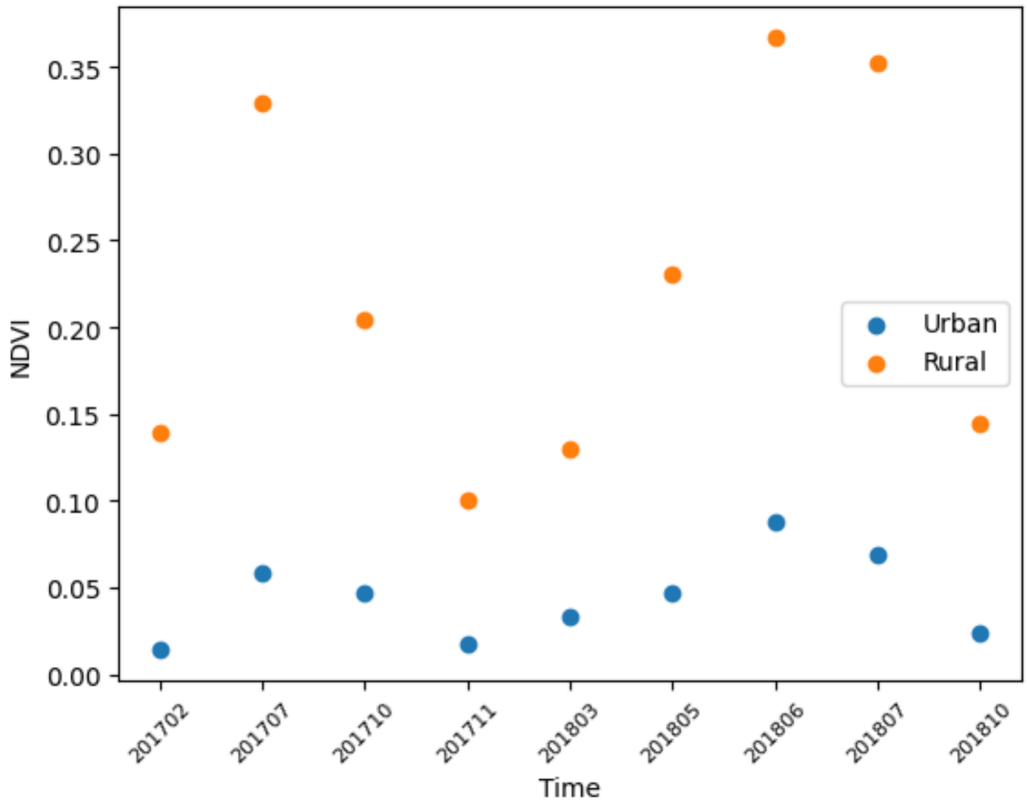
```
# Add x-tick labels
mon_ticks = ['J', 'F', 'M', 'A', 'M', 'J', 'J', 'A', 'S', 'O',' N',' D']
plt.xticks(x, mon_ticks)
plt.legend()

# Plot precipitation timeseries
plt.figure(2)
width = 0.35
plt.bar(x - width/2, Prec_gbg_2017_2018_mon_mean, width, label='2017-2018')
plt.bar(x + width/2, Prec_gbg_1981_2010_mon_mean, width, label='1981-2010')

# Labels
plt.xlabel('Time [months]')
plt.ylabel('Monthly mean precipitation [mm]')

# Add x-tick labels
mon_ticks = ['J', 'F', 'M', 'A', 'M', 'J', 'J', 'A', 'S', 'O',' N',' D']
plt.xticks(x, mon_ticks)
plt.legend()

plt.figure(3)
plt.bar(np.arange(Prec_gbg_2017_2018.precip.values.__len__()),␣
 ↪Prec_gbg_2017_2018.precip.values)
plt.xticks(np.arange(Prec_gbg_2017_2018.precip.values.__len__()), mon_ticks*2)
plt.xlabel('Time [months]')
plt.ylabel('Precipitation [mm]')

plt.figure(4)

result_date = np.int_(result[:,0])

plt.scatter(np.arange(result.shape[0]), result[:,1], label='Urban')
plt.scatter(np.arange(result.shape[0]), result[:,2], label='Rural')
plt.legend()

plt.xticks(np.arange(result.shape[0]), result_date.astype(str), rotation=45,␣
 ↪fontsize=8)

plt.xlabel('Time')
plt.ylabel('NDVI')
```

[9]: Text(0, 0.5, 'NDVI')


2017-2018
1981-2010
Monthly mean temperature [K]
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
J
F
M
A
M
J
J
A
S
O
N
D
Time [months]


2017-2018
1981-2010
Monthly mean precipitation [mm]
0
20
40
60
80
J
F
M
A
M
J
J
A
S
O
N
D
Time [months]

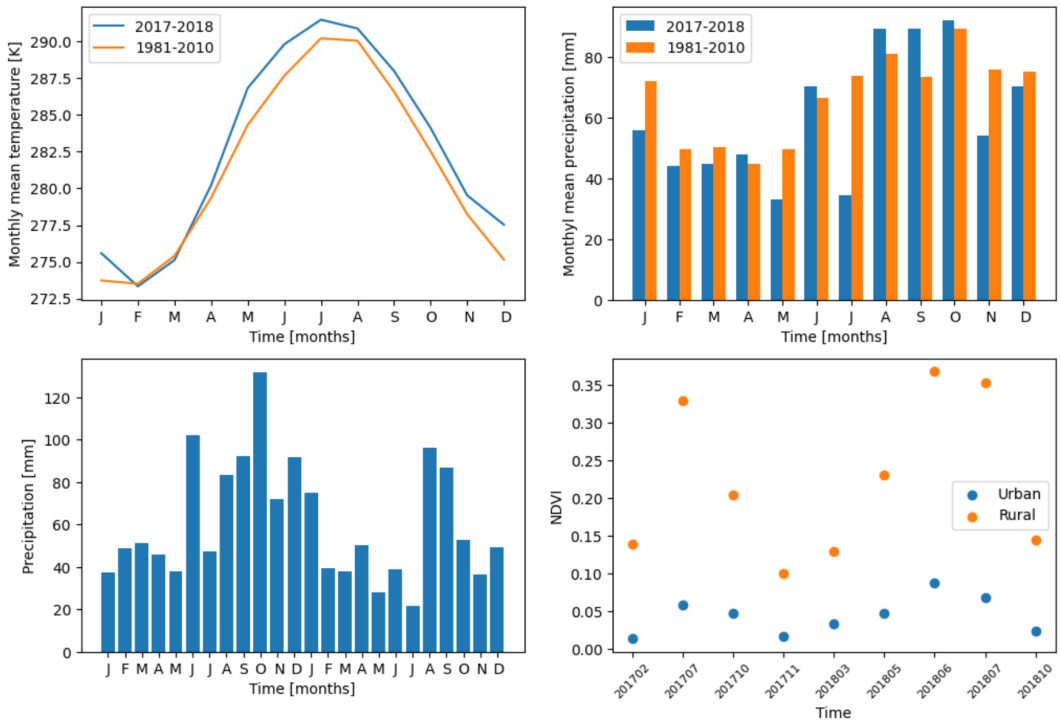

Precipitation [mm]
0
20
40
60
80
100
120
J F M A M J J A S O N D J F M A M J J A S O N D
Time [months]



[10]:
```
# Now we want to plot the data above

# PLOTTING
fig, axs = plt.subplots(2, 2, figsize = (12,8))

'''Figure 1'''
# Plot air temperature timeseries
x = np.arange(Tair_gbg_1981_2010_mon_mean.__len__())
axs[0,0].plot(x, Tair_gbg_2017_2018_mon_mean, label='2017-2018')
axs[0,0].plot(x, Tair_gbg_1981_2010_mon_mean, label='1981-2010')

# Labels
axs[0,0].set(xlabel='Time [months]', ylabel='Monthly mean temperature [K]')

# Add x-tick labels
mon_ticks = ['J', 'F', 'M', 'A', 'M', 'J', 'J', 'A', 'S', 'O',' N',' D']
axs[0,0].set_xticks(x, mon_ticks)
axs[0,0].legend()

'''Figure 2'''
```

```
# Plot precipitation timeseries
width = 0.35
axs[0,1].bar(x - width/2, Prec_gbg_2017_2018_mon_mean, width, label='2017-2018')
axs[0,1].bar(x + width/2, Prec_gbg_1981_2010_mon_mean, width, label='1981-2010')

# Labels
axs[0,1].set(xlabel='Time [months]', ylabel='Monthyl mean precipitation [mm]')

# Add x-tick labels
mon_ticks = ['J', 'F', 'M', 'A', 'M', 'J', 'J', 'A', 'S', 'O',' N',' D']
axs[0,1].set_xticks(x, mon_ticks)
axs[0,1].legend()

'''Figure 3'''
# Plot precipitation from January 1st 2017 - December 31st 2018
axs[1,0].bar(np.arange(Prec_gbg_2017_2018.precip.values.__len__()),␣
 ↪Prec_gbg_2017_2018.precip.values)
axs[1,0].set_xticks(np.arange(Prec_gbg_2017_2018.precip.values.__len__()),␣
 ↪mon_ticks*2)
axs[1,0].set(xlabel='Time [months]', ylabel='Precipitation [mm]')

'''Figure 4'''
# Plot NDVI
result_date = np.int_(result[:,0])

axs[1,1].scatter(np.arange(result.shape[0]), result[:,1], label='Urban')
axs[1,1].scatter(np.arange(result.shape[0]), result[:,2], label='Rural')
axs[1,1].legend()

axs[1,1].set_xticks(np.arange(result.shape[0]), result_date.astype(str),␣
 ↪rotation=45, fontsize=8)

axs[1,1].set(xlabel='Time', ylabel='NDVI')
```

[10]: [Text(0.5, 0, 'Time'), Text(0, 0.5, 'NDVI')]



Finished! Great! Let's do it for Luleå now. You find the necessary Landsat data on Canvas.

Coordinates for Luleå bounding box are:
lonmin = 530000 # Four zeros
latmin = 7250000 # Four zeros
lonmax = 570000 # Four zeros
latmax = 7300000 # Four zeros

Coordinates:
latlul = 65.584
lonlul = 22.155