

Technical Report

F. Vrolijk
FutureWater
Supervisors: A. Lutz and H. Biemans

November 20, 2016



Contents

1	Introduction to Report	3
2	Climate Data Operators (CDO)	3
2.1	Preparation netCDF File	3
2.2	Consecutive Dry Days	4
2.3	Number of Heat Waves	4
3	Reprojection	6
3.1	Load Packages	6
3.2	Read netCDF	6
3.3	Reprojection	6
3.4	Plot Using Basemap	7
4	Dictionary	8
4.1	Benefits of Using a Dictionary for Plotting	8
4.2	Defining a Dictionary	8
5	Read Data from Dictionary	9
6	Map output	10
6.1	Maps of Daily Precipitation and Temperature Values – Preparation of the Data	10
6.2	Iteration to Create Daily Output Values with Basemap	11
6.3	Read Data and Scale Bar Range from Dictionary	11
6.4	Write Output	12
7	Clip Raster Using GDAL	13
7.1	Dictionary with district extents	13
7.2	Clip Raster Files Using GDAL Translate	13
8	Zonal Stats	15
8.1	Prepare Data	15
8.2	Use Python to Run Algorithm from QGIS Toolbox	16
8.3	Write Output to CSV	16

1 Introduction to Report

The internship at FutureWater mostly focussed on scripting challenges in order to produce maps and CSV output. Therefore, in addition to the main internship report, this technical report aims to demonstrate some of the coding challenges that were encountered during the internship period. Consequently, this report describes what types of tasks were dealt with, and which Python tools were used during the project. Lastly, this document shows the approach that was used to solve the coding challenges.

This report shows several important steps that make up part of the code that was used to get the output, as shown in the main report. The operations are divided into seven components that are ordered in such a way that all described operations create a coherent overview of (a part of) the written code.

2 Climate Data Operators (CDO)

The [Climate Data Operators](#) from the Max Planck Institute are a collection of powerful operators for handling netCDF files and perform calculations with climate datasets. For the calculation of climate indices, many different functions have been created by the institute that perform one specific task, such as calculating the yearly mean precipitation or the maximum temperature. Other functions are aimed at calculating indices of temperature and precipitation extremes. These functions were used to perform the calculations in the IGB basin. Two such examples are shown in this chapter; one function to calculate the number of consecutive dry days (Chapter 2.2) and another function to calculate the number of heat waves (Chapter 2.3).

2.1 Preparation netCDF File

First, import the required packages to extract information from the netCDF files.

```
# Import modules
from cdo import *
cdo = Cdo()
import numpy as np
from netCDF4 import Dataset
from os.path import expanduser
```

Next, the home directory for the input files can be set using the “expanduser” function

```
# Set home directory input files
ref_files =
expanduser("/Volumes/Naamloos/climate_data/reference/10km/merged_files/")

# Set input files
infile_prec = ref_files+'/prec.nc'
infile_avg_temp = ref_files+'/avg_temp.nc'
infile_max_temp = ref_files+'/max_temp.nc'
```

```
# Select input file
input_file = "/Volumes/Naamloos/climate_data/final_files/prec_mean.nc"
```

2.2 Consecutive Dry Days

The function shown below is used to calculate the number of consecutive dry days for a given netCDF input file. The user can enter a threshold (using the “raw_input” functionality) in millimetres to count the number of consecutive days below the provided threshold (default is 1 mm). The output file contains the name of calculated index, along with the minimum number of days and precipitation threshold for a dry period.

```
def consecutive_dry_days():
    """
    Count the number of consecutive days where the amount of precipitation
    is less than a certain threshold.
    Set the number of consecutive days below the threshold. Precipitation
    threshold, unit: mm (default: 1 mm);
    Minimum number of days exceeding the threshold.

    :return: Number of dry periods of more than [nr_of_days] with a
    threshold of [prec_threshold]
    """

    # Define output directory and ask user for input for CDO function
    ofile_cdd = '/Volumes/Naamloos/climate_data/reference/10km/output/cdd_'
    inputs = ('mm', 'days', '.nc')
    input_mm = raw_input('Enter the threshold in [mm] for calculating the
    periods with dry days:')

    # CDO function to calculate the number of consecutive dry days
    cdo.eca_cdd(input_mm, input=ifile_prec,
    output=ofile_cdd + input_mm + '{0}'.format(*inputs) +
    '{2}'.format(*inputs))
```

2.3 Number of Heat Waves

Similarly, the number of heat waves in a certain period can be calculated with “heat_wave” function. The user can define the minimum number of days and temperature threshold to calculate the number of heat waves.

```
def heat_wave():
    """
    Calculate the number of periods in a time series. This is calculated by
    comparing a reference temperature against the maximum temperature.
    Counted are the number of periods with at least [n] consecutive days
    and a temperature offset [t]. Temperature offset default is 5 C, days
    is 6.

    :return: Number of periods with [n] consecutive days and [t]
    temperature offset from the mean.
    """
```

```

# Define output directory and ask user for input for CDO function
ofile_hwdi =
'/Volumes/Naamloos/climate_data/reference/10km/output/hwdi_'
inputs = ('C', 'days', '.nc')
input_temp = raw_input('Enter the offset of temperature in [degrees C]
for calculating the periods with a heat wave:')
input_days = raw_input('Enter the minimum number of consecutive days
for a heat wave:')

# CDO function to calculate the number of heat wave
cdo.eca_hwdi(input_days, input_temp, input=" ".join((ifile_max_temp,
ifile_avg_temp)),
output=ofile_hwdi + input_temp + '{0}'.format(*inputs) + '_' +
input_days + '{1}'.format(*inputs) + '{2}'.format(*inputs))

```

3 Reprojection

The final output from the CDO functions is ready to be displayed in QGIS or other GIS software. However, when dealing with many output products, plotting of the output can become a tedious task. Many different output products were created during the project. Therefore, the process of plotting was automated as much as possible. The “Basemap” package from the extensive “matplotlib” Python library was used to plot the output on a basemap.

3.1 Load Packages

First, load the required Python packages for plotting and reprojecting with Basemap.

```
# Import modules
import mpl_toolkits.basemap.pyproj as pyproj
from netCDF4 import Dataset
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

3.2 Read netCDF

Once an input file is selected, it can be read using the “Dataset” function from the netCDF4 package to obtain information about the file. The example shown below contains yearly precipitation data (from 1981 to 2010) for the entire IGB basin.

```
# Read netCDF
file = 'prec_mean_1981-2010_yearly.nc'
file_handle = Dataset(file, mode='r')
lon = file_handle.variables['lon'][:] # 320
lat = file_handle.variables['lat'][:] # 190
var = file_handle.variables['P'][:] # (30, 320, 190)
file_handle.close()
```

3.3 Reprojection

The basemap and netCDF file are not yet in the same projection. The “pyproj” function was used to project from the WGS 84 / UTM Zone 45N to WGS 84 projection. The function input can simply be given as an EPSG-code (as used in “Source projection”) or a “Proj4” string (as used in “Target projection”), obtained from a [spatial reference website](#).

```
# Source projection
UTM_45N = pyproj.Proj("+init=EPSG:32645")

# Target projection
WGS84 = pyproj.Proj("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
```

Before the actual transformation can be performed, the 1D arrays containing the longitudes and latitudes have to be modified into 2D arrays. The “meshgrid” function

from the “Numpy” package is used to create these 2D arrays. These arrays are used to perform the transformation.

```
# Transformation
lons, lats = np.meshgrid(lon, lat)
tr_lon, tr_lat = pyproj.transform(UTM_45N, WGS84, lons, lats)
```

3.4 Plot Using Basemap

Finally, the [Basemap](#) library is used to define the location and extent of the study area. This background map can easily be adapted and elaborated by adding country boundaries, coastlines or a satellite background.

```
# Basemap
plt.figure()
m = Basemap(projection='tmerc', resolution='l', height=3000000,
width=4000000, lat_0=27.5, lon_0=82.5)
m.shadedrelief()
xi, yi = m(tr_lon, tr_lat)
cs = m.pcolormesh(xi, yi, pr[0], cmap='YlGnBu')
plt.show()
```

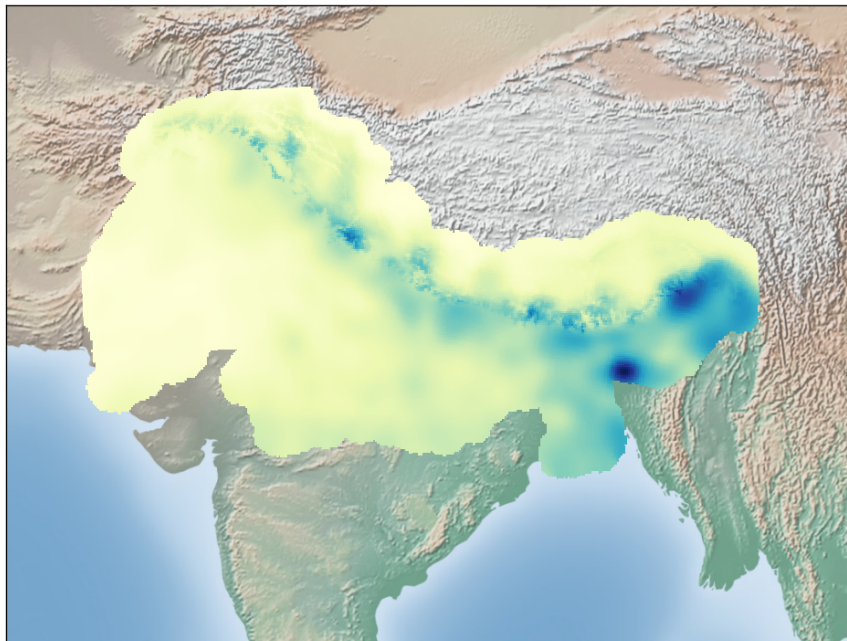


Figure 1: Example output of yearly mean precipitation with background image.

4 Dictionary

4.1 Benefits of Using a Dictionary for Plotting

Dictionaries can be thought of as unordered key – value pairs that can store information. The HI-AWARE project requires many different output maps of precipitation and temperature extremes. To automate the process of creating maps for these different output files, a dictionary can be used to efficiently store information for plotting purposes.

Keys are unique within a dictionary. For each output map, a corresponding key is created in the dictionary that points to the data file. In addition, one unique key contains the variable name of the file, along with the title and scalebar text for the output map.

4.2 Defining a Dictionary

An example of a dictionary used for this project is displayed below, showing several key – value pairs, where each key in the dictionary points to a unique file. The title that is stored in the key of a dictionary is almost complete. However, it is still missing the district name for the different areas in the study area. This name is added at the end of each value, for each key in the dictionary. This makes it possible to iterate over the districts and subsequently print the full title names to the map.

```
# Define dictionary containing the netCDF file name [0], along with the
variable name [1], scale bar text [2] and map title [3]
dictionary = {
'prec_mean': ['prec_mean_1981-2010_yearly.nc', 'P', 'Yearly mean precipitation
(mm)', 'Mean precipitation in '],

'temp_mean': ['temp_mean_1981-2010_yearly.nc', 'Tavg', 'Mean temperature
(degrees Celsius)', 'Yearly mean temperature in '],

'temp_max': ['temp_max_1981-2010_yearly.nc', 'Tmax', 'Maximum temperature
(degrees Celsius)', 'Yearly maximum temperature in '],

'temp_min': ['temp_min_1981-2010_yearly.nc', 'Tmin', 'Minimum temperature
(degrees Celsius)', 'Yearly minimum temperature in '],

'cwd_cwdi_per_time_period': ['cwd_10mm.nc',
'consecutive_wet_days_index_per_time_period', 'Consecutive wet days index
(CWDI)', 'Greatest number of consecutive days with daily precipitation \n
above 10 mm between 1981 and 2010 '],

'cdd_cddi_per_time_period': ['cdd_1mm.nc',
'consecutive_dry_days_index_per_time_period', 'Consecutive dry days index
(CDDI)', 'Greatest number of consecutive days with daily precipitation \n
below 1 mm between 1981 and 2010 '],

}
```


5 Read Data from Dictionary

Once a dictionary is constructed, it can be used to extract information for plotting purposes. Before a plotting function can use the data from a dictionary, it has to be extracted. The function in this chapter achieves this by reading the data from a key, which is linked to a file. The function shown below reads in a file and extracts the variable name from the input file. In addition, it reads the minimum and maximum value for this variable (e.g. precipitation in mm). The extracted data from the variable is used to create a map. The minimum and maximum value of the variable are used to define the range of the scalebar when the data is plotting with the Basemap package. This guarantees that the entire range of values is always covered, regardless of the type of data that is plotted.

```
import numpy as np
from netCDF4 import Dataset

def read_data(dictionary, key):
    """
    Reads data from a dictionary containing the file names and variable
    names of netCDF files
    :param dictionary: Dictionary containing the file names and variable
    names of each output file
    :param key: Key to be used to read data regarding variable information
    :return: Data of the (var), the minimum value (min_value) and maximum
    value (max_value) for the input file
    """
    file_name = dictionary[key][0]
    variable = dictionary[key][1]
    path = '/Volumes/Naamloos/climate_data/final_files/'
    data = path + file_name
    file_handle = Dataset(data, mode='r')
    var = file_handle.variables[variable][:]
    min_value = np.amin(var)
    max_value = np.amax(var)
    return var, min_value, max_value

read_scalebar(dictionary, 'prec_mean')
```

6 Map output

After preparation of the data, the reprojection and the creation of the dictionary, the data can be plotted. Three main types of map output were created: (i) maps of precipitation and temperature extremes, (ii) maps of yearly values (e.g. mean precipitation) and (iii) maps with daily output (e.g. maximum temperature). Three functions were created to process the data and create the desired output. This chapter shows one of these functions: creating daily outputs.

6.1 Maps of Daily Precipitation and Temperature Values – Preparation of the Data

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

def daily_outputs(dictionary, key, cs, nr_of_days):
    """
    Create map with daily output for precipitation and temperature
    :param dictionary: Dictionary containing information about file name,
    variable name, map title and scalebar text
    :param key: The key in the dictionary pointing to the file that should
    be used
    :param cs: The colour scheme to be used for the output maps, e.g.
    'Blues' or 'Oranges'
    :param nr_of_days: Number of maps (one map for each day) that the
    function should return
    :return: One map for each day, showing daily precipitation or
    temperature values
    """
```

After reading the required packages into Python, the data from the dictionary can be extracted. The scale bar text is also obtained from the dictionary, which will be used to add a description to the scale bar (e.g. the units of measurement).

```
# Read data from dictionary
datas = read_data(dictionary, key)[0]
scalebar = dictionary[key][2]
```

For each map, a date has to be printed in the title. The “date_range” function from the “Pandas” package can be used to create a list of dates, in this case ranging from the first of January 1981 to December 31st of 2010.

```
# Define the date range that should be used for the output maps
days = pd.date_range(start='1980-12-31', end='2010-12-31', freq='D',
closed='right')
days = pd.Series(days.format())
```

```
# User defined data type
type = raw_input('Which type of data should be used? (e.g. mean
precipitation, maximum temperature)')
```

6.2 Iteration to Create Daily Output Values with Basemap

After the user has defined the dictionary, key and colour scheme to be used, the function can iterate over the number of days that are specified in the function input. For each iteration a basemap background map will be created that uses a Transverse Mercator projection and the extent of the entire IGB basin. For plotting other districts within the basin, “if” statements are additionally used in the function to determine which basemap should be plotted.

```
for i in range(0, nr_of_days):
    # Define the base map projection, extent and appearance
    m = Basemap(projection='tmerc', resolution='l', height=2300000,
width=3500000, lat_0=29, lon_0=82)
    m.drawmapboundary()
    m.drawparallels(np.arange(-80., 81., 10.), labels=[1, 0, 0, 0])
    m.drawmeridians(np.arange(-180., 181., 10.), labels=[0, 0, 0, 1])
    m.drawcountries()
    m.shadedrelief()
```

6.3 Read Data and Scale Bar Range from Dictionary

Next, the netCDF data can be plotted over the background map. This process starts with reading the coordinates for plotting. “Pcolormesh” is used to plot the data of each iteration over the basemap, and uses the user-defined colorscheme. Subsequently, “colorbar” is used to create the accompanying scalebar of the map, using the scale bar text from the dictionary. Furthermore, the minimum and maximum values of the netCDF data are read through the “read_data” function, which extracts the minimum and maximum values from the data arrays. Finally, “clim” is used to integrate these values during the plotting process.

```
# Obtain the latitudes and longitudes, point to data and read the
minimum and maximum values for the scalebar
xi, yi = m(tr_lon, tr_lat)
netcdf = m.pcolormesh(xi, yi, datas[i], cmap=cs)
m.colorbar(netcdf, location='bottom', pad='10%', label=scalebar)
sb_min = read_data(dictionary, key)[1]
sb_max = read_data(dictionary, key)[2]
plt.clim(sb_min, sb_max)
```

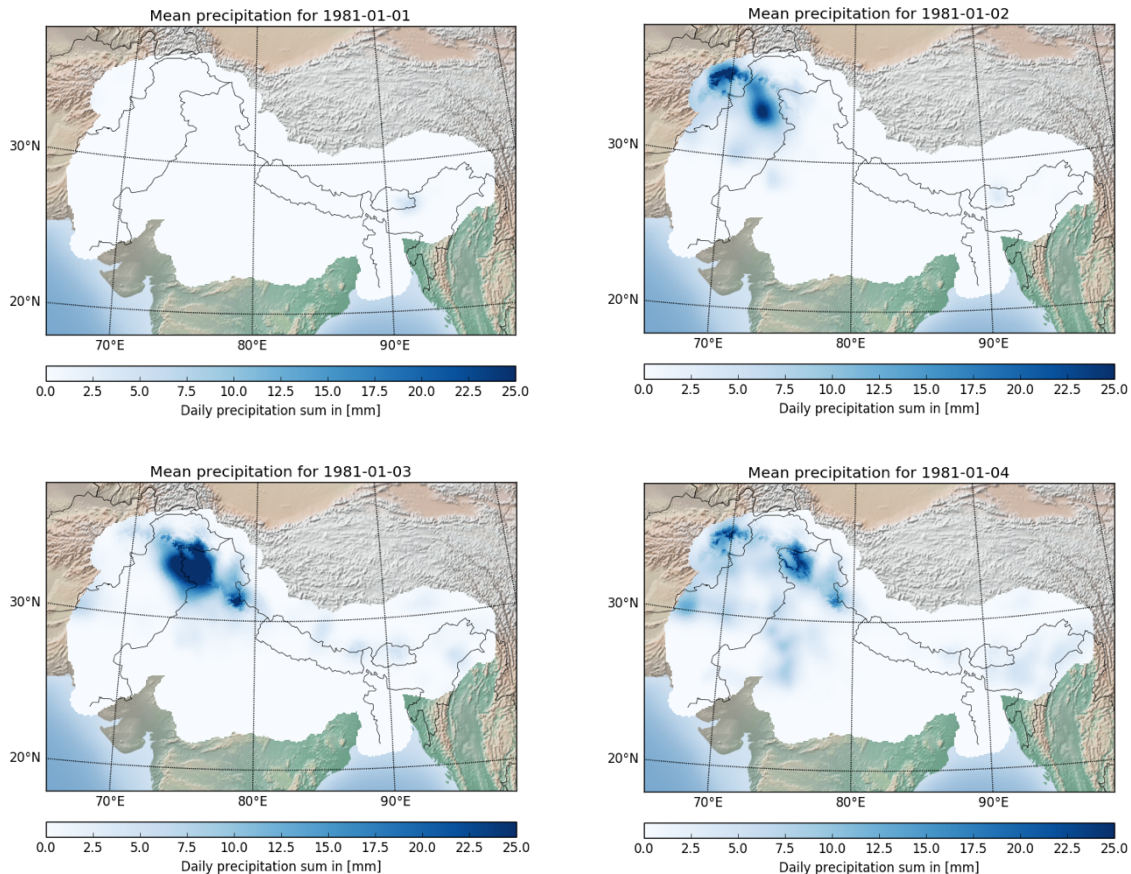


Figure 2: Example output of daily precipitation from January first to January fourth, 1981.

6.4 Write Output

The map is now almost ready to be saved to disk. All that remains is creating an appropriate map title and file name. These should include the specific day for which the map was created. This is achieved by using the “format” functionality that allows user input or values that are stored in a variable to be “glued” together into one appropriate title or file name.

```
# Define location for map output and create filename and title for the map
path = '/users/froedevrolijk/Geo/outputs_hiaware/entire_IGB/daily'
f_name = dictionary[key][1]+'{0}.png'.format(days[0:nr_of_days][i])
plt.title(type + ' for {0}'.format(days[0:nr_of_days][i]))
filename = os.path.join(path, f_name)
```

Before the map is saved to disk, the function checks if the file already exists. If so, it deletes the old file before it saves the map, and finally closes the plotting window.

```
# If the output file already exists, remove the old file
if os.path.isfile(filename):
    os.remove(filename)

# Save the map, and close the plotting window
plt.savefig(filename)
plt.close()
```

7 Clip Raster Using GDAL

The function that creates the final maps the was shown in the previous chapter, and chapter 5 described the function to derive the minimum and maximum values of the data, in order to plot the entire range of values in the map. However, for the districts (Chitwan, Faisalbad, Gandaki, Gilgit and Pothwar) only a small proportion of the map is shown. Therefore, also the range of values for a certain variable often varies substantially from one location to the other. The full scale range can therefore not be used for each district, as the colourscheme would not be appropriate for the region of interest.

One solution to this problem is to clip each output file used for plotting to a new extent. This guarantees that only the values proximate to the district area are considered for the colour scheme and scale bar. For this project, many output files were clipped to a new extent (~400 files). This was achieved using GDAL Translate from a Python script, which iterates over a folder with files and executes a piece of code in the command line (bash).

7.1 Dictionary with district extents

The functionality to send code to the command line from within Python is part of the “subprocess” package. First, a dictionary with the extents of the districts is constructed. The keys in the dictionary are used to iterate over the coordinates and clip each output file accordingly.

```
import os
import glob
import subprocess

extents_dict = {'chitwan': [166058.87, 3116189.07, 312721.71, 2998326.01],
                 'faisalbad': [-915605.83, 3641383.32, -732202.73, 3441863.91],
                 'gandaki': [-3146.22, 3347916.98, 480904.11, 2927078.34],
                 'gilgit': [-851712.00, 4209257.48, -564539.79, 3986650.86],
                 'pothwar': [-1035816.99, 3958256.70, -642195.94, 3578737.26]}
```

7.2 Clip Raster Files Using GDAL Translate

The function input only requires the dictionary with the extents as input. After defining an input and output folder, the files can be clipped to their new extent.

```
def clip_rasters(extents_dict):
    """
    Function to clip netCDF files to a new extent as used for the various
    districts.
    :param extents_dict: The coordinates of the districts in a dictionary,
    formatted as ulx uly lrx lry
    :return: Clipped files for each key in the dictionary
    """

    # Define input and output folder
```

```

dfolder = "/Volumes/Naamloos/climate_data/final_files/"
ofolder = "/Volumes/Naamloos/climate_data/final_files/ClipRaster/"
# Remove xml files from input folder
xml_files = glob.glob(dfolder + '*.xml')
[os.remove(xml) for xml in xml_files]

```

A nested loop is used to clip the files. The first loop iterates over the extent keys in the dictionary and derives the coordinates that should be used with GDAL Translate. The second loop is used to iterate over the keys in the original dictionary, which contains the file and variable names for each file (“ncfile” and “vars” respectively, as shown in the code). As a result, each key is clipped to the extents in the “extents_dict” dictionary, i.e. for all districts included in the dictionary. More districts can easily be added to the dictionary, to automate the process of clipping the files in a folder.

```

# Get the coordinates for each district
for extent_key in extents_dict:
    coordinates = extents_dict.get(extent_key)

# Clip each file in the dictionary by calling gdal translate via
subprocess
for key in dictionary:
    indexes = dictionary[key]
    ncfile = indexes[0]
    vars = indexes[1]

```

What remains is the GDAL Translate code, which is sent to the command line using the “subprocess” functionality. The filename includes the key name of each file, and adds the district name to the output file.

```

# Call GDAL translate via subprocess to execute the command in
bash
subprocess.call('gdal_translate -of netCDF -projwin' + ' ' +
str(coordinates[0]) + ' ' + str(coordinates[1]) + ' ' + str(
coordinates[2]) + ' ' + str(coordinates[3]) + ' NETCDF:' +
ncfile + ':' + vars + '" ' + ofolder + key + '_' + extent_key +
'_clip.tif', shell=True, cwd=dfolder)

print 'All done!'

```

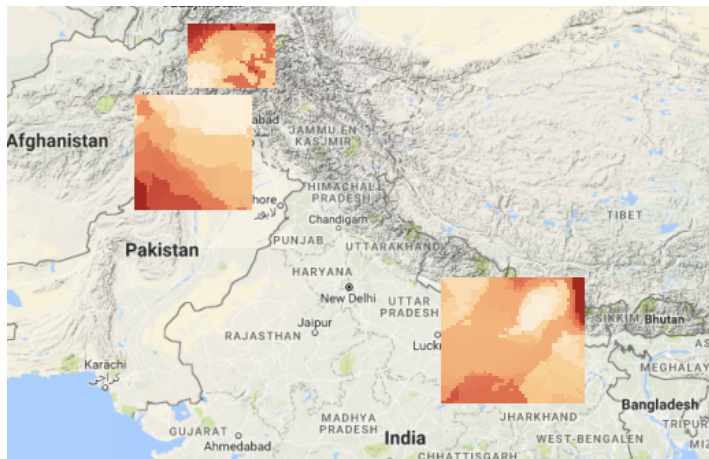


Figure 3: Example output of clipped files.

8 Zonal Stats

In addition to the map output, a quantitative output is created for each file in the form of CSV output. For example, yearly mean precipitation values for 30 years, which are displayed in a single CSV output file. This is achieved using a SAGA algorithm from the QGIS Toolbox. This chapter discusses some of the code that is used to create this type of output.

8.1 Prepare Data

The input into the function is a dictionary with extents of the districts. The function returns a single CSV file for each netCDF data file, and each district.

```
import os
import glob
import shutil
import processing
import pandas as pd

# Zonal statistics
def zonal_stats(extents_dict):
    """
    Calculate zonal statistics for a number of districts, using coordinates
    from a dictionary.
    :param extents_dict: Dictionary containing the coordinates of the
    districts
    :return: CSV output for each input file containing zonal statistical
    values for each time step
    """

    # Download folder for output
    dfolder = "/Volumes/Naamloos/climate_data/final_files/ZonalStats/"
```

The preparation process includes the creation of an output folder for each district, if it does not yet exist. Any old files in these folders are deleted, if necessary.

```
# Create folders per district for output, and remove old files if
    necessary
    try:
        if not os.path.exists(os.path.join(dfolder +
            str(extents_dict.keys))):
            [os.mkdir(os.path.join(dfolder + extents_key)) for extents_key
            in extents_dict]

    except:
        output_files =
        glob.glob('/Volumes/Naamloos/climate_data/final_files/ZonalStats/*')

        [shutil.rmtree(output_file, ignore_errors=True) for output_file in
        output_files]
```


The SAGA algorithm “gridstatisticsforpolygons” uses the shapefiles of the districts and the raster files to calculate zonal statistics. The required files are listed using the “glob.glob” functionality.

```
# List shapefiles to calculate statistics
shp_path = '/froedevrolijk/Dropbox/Tailoring_climate_data/data/proj_shp/'
shps = glob.glob(shp_path + '*.shp')

# List netCDF files
local = os.path.expanduser('/Volumes/Naamloos/climate_data/final_files/')
netcdf_path = glob.glob(local + '*.nc')

# Find files in clipped directory
clipped_files_path = os.path.join(local + 'ClipRaster/')
clipped_files = glob.glob(clipped_files_path + '*.tif')

# Remove xml files
xml_files = glob.glob(dfolder + '*.xml')
[os.remove(xml) for xml in xml_files]
```

The clipped files in the folder have the district names included in their file name. The next step is to list the files for each district so that they can be used to run the algorithm for a certain district.

```
# Split files into district groups and create a list of the districts
gandaki = [c for c in clipped_files if "gandaki" in c]
faisalbad = [c for c in clipped_files if "faisalbad" in c]
chitwan = [c for c in clipped_files if "chitwan" in c]
pothwar = [c for c in clipped_files if "pothwar" in c]
gilgit = [c for c in clipped_files if "gilgit" in c]
```

8.2 Use Python to Run Algorithm from QGIS Toolbox

For each raster file in a district folder, the [SAGA algorithm](#) can be executed (iteratively) in QGIS. An example of this is shown below.

```
# For tif in gandaki:
processing.runalg("saga:gridstatisticsforpolygons", tif, gandaki_shp,
False, False, False, False, False, True, False, False, 0, dfolder +
'cdd_cddi_per_time_period_chitwan_clip_b1.csv'[tif])
```

The district lists can be transformed into a list of lists, containing the district names, where each district list contains the raster files to calculate the zonal statistics.

8.3 Write Output to CSV

Each timestep in a file will output only one value in a CSV file. The number of timesteps is linked to the number of bands in a file. For example, a file containing 30 years of yearly precipitation values contains 30 bands – and each band will output one value in a single CSV file. It is therefore necessary to adapt the output, in order to write these 30 values to a single CSV file for each netCDF file.

First, the path to the CSV output is defined. The CSV output files are listed for each raster file of a certain district.

```
# Write csv output for each district
for district_files in district_list:
    district = district_names[district_list.index(district_files)]

    # List csv files in output directory
    csv_files_path = os.path.join(local + 'ZonalStats/')

    # Loop through the tif files of each district to get csv files (linked
    # to the number of years)
    for tif_file in district_files:
        csv_files = glob.glob(csv_files_path + '*' +
                               tif_file.rstrip('.tif').split('/')[1] + '*.csv')
```

ISO	NAME_0	ID_1	NAME_1	ID_2	NAME_2	ID_3	NAME_3	CCN_3	VolumesNaam
NPL	Nepal	1	Central	3	Narayani	16	Chitawan	0	2944,977273

ISO	NAME_0	ID_1	NAME_1	ID_2	NAME_2	ID_3	NAME_3	CCN_3	cddnrddper
NPL	Nepal	1	Central	3	Narayani	16	Chitawan	0	270,3181818

ISO	NAME_0	ID_1	NAME_1	ID_2	NAME_2	ID_3	NAME_3	CCN_3	VolumesNaam
NPL	Nepal	1	Central	3	Narayani	16	Chitawan	0	88,22727273

Figure 4: Example yearly CSV output (three years) for the number of consecutive dry days in the Chitwan district.

The function checks if the CSV file has any value, before it is copied. Next, a newly created CSV file is filled with 0's, which can subsequently be filled with values from the CSV files. Each row in the new CSV file represents a timestep (e.g. a yearly value).

```
# Check if output is generated for the csv file
if len(csv_files) > 0:
    summed_data = pd.DataFrame(columns=['Value'], index=range(1,
        len(csv_files) + 1))
    summed_data = summed_data.fillna(0.0)

    # Retrieve output value from csv files
    row = 0
    for csv in csv_files:
        row += 1
        csv = pd.read_csv(csv, header=0)
        cleaned = csv.iloc[:, -1]

        summed_data.loc[row, ] = round(float(cleaned), ndigits=1)
    print(summed_data)
```

Year	Value
2011	2945,0
2012	270,3
2013	88,2

Figure 5: Merged output with yearly values for the number of consecutive dry days.

Lastly, an output folder for the CSV files is created, if it does not exist yet. The CSV files can be saved in their appropriate output folders.

```
# Create folders for output
try:
    if not os.path.exists(os.path.join(csv_files_path +
        str(district))):
        os.mkdir(os.path.join(csv_files_path + str(district)))
except:
    pass

summed_data.to_csv(csv_files_path + str(district) + '/' +
tif_file.rstrip('.tif').split('/')[1] + '.csv')
```