Short Course 6.3

Introduction to Python for Earth System Sciences

Bidyut Bikash Goswami
Institute of Science and Technology Austria
Klosterneuburg, Austria

Note: Following slides contain screenshots of the code already shared on Github

```
#To plot seasonal mean rainfall for a region
#Ideantify EXCESS and DEFICIT years (defined as above or below 1SD from mean, respectively)
```

Task: 1

1. Read data year by year
2. Select (slice) the duration (time) and region (longitude and latitude) of interest
3. Compute average of the selected data in step 2
4. Store averagred data in one time series
5. plot the data
6. Beautify the plot

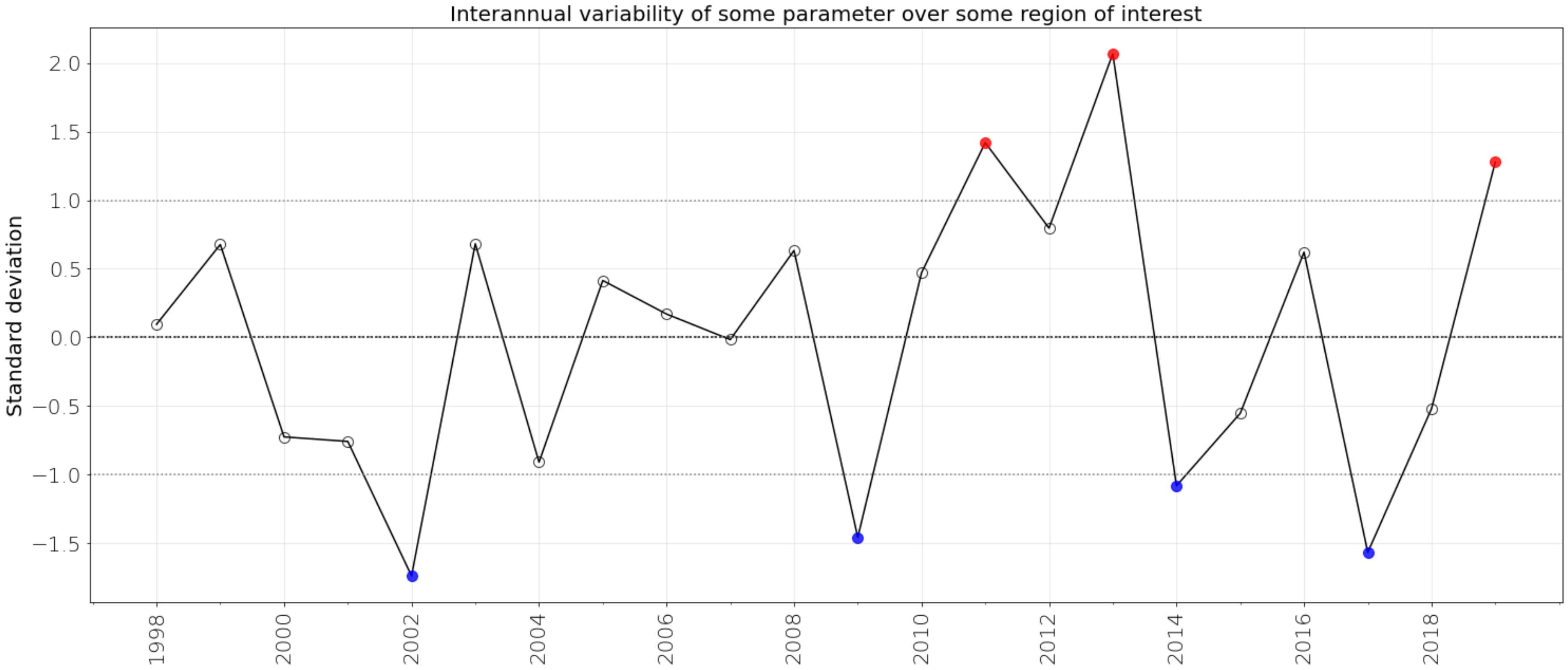
Task: 2

7. Compute SD of the time series
8. Identify excess/flood and deficit/drought years
9. Mark the excess and deficit years in the data.

Task: 3 (optional)

10. Store the Excess(flood) and Deficit(drought) years
11. What is the structure of acssociated spatial structure of another data.

Target plot


Interannual variability of some parameter over some region of interest
Standard deviation
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018

To do numerical operations

To visualize data

To open data

To access list data for opening

To see progress bar (optional)

To select data in time axis

To “ignore” warnings (optional)

```
#importing necessary packages

import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import glob
from tqdm.notebook import tqdm
from datetime import datetime, timedelta

#Package to suppress Python warnings
import warnings
warnings.filterwarnings("ignore")
```

```python
#BEFORE ANALYSING THE DATA

#Study specifications

#Declaring start year and end year of my analysis
start_year = 1998
end_year   = 2019
end_year_idx = end_year+1

#Region boundaries
west_lon = 75.0
east_lon = 84.0
south_lat= 18.0
north_lat= 28.0
```

Because Python counts from 0

```
#A look at the data
#Note: We can use terminal comands in Jupyter notebook by using "!" sign
!ncdump -h Path to data/TRMM_daily_0.25x0.25_1998.nc
```

```
netcdf TRMM_daily_0.25x0.25_1998 {
dimensions:
	time = UNLIMITED ; // (365 currently)
	lon = 1440 ;
	lat = 400 ;
variables:
	double time(time) ;
		time:standard_name = "time" ;
		time:units = "hours since 1-1-1 00:00:00" ;
		time:calendar = "standard" ;
		time:axis = "T" ;
	double lon(lon) ;
		lon:standard_name = "longitude" ;
		lon:long_name = "longitude" ;
		lon:units = "degrees_east" ;
		lon:axis = "X" ;
	double lat(lat) ;
		lat:standard_name = "latitude" ;
		lat:long_name = "latitude" ;
		lat:units = "degrees_north" ;
		lat:axis = "Y" ;
	float r(time, lat, lon) ;
		r:long_name = "Daily accumulated precipitation (combined microwave-IR) estimate with gauge calibration over lan
d  [mm]" ;
		r:_FillValue = -9999.9f ;
		r:missing_value = -9999.9f ;
```

Its TRMM3b42 version 7 0.25x0.25 degree gridded data listed yearly

```python
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

To see progress bar (optional)

```python
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('[redacted]/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

Path to data

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
range(start_year, end_year_idx)
```

```
range(1998, 2020)
```

```
for i in range(start_year, end_year_idx):
    print (i)
```

```
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
```

```python
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/          Any path          /*')
```

Same as doing “ls”

To see progress bar (optional)

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('          Path to data          /TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/ Any path /*')
```

Same as doing "ls"

To see progress bar (optional)

Reading file name

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/ Any path /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/          Any path          /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('          Path to data          /TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

```
100% tqdm 22/22 [02:23<00:00, 7.88s/it]
```

```
datetime.strptime("01 Jun, 2000", "%d %b, %Y")
```

```
datetime.datetime(2000, 6, 1, 0, 0)
```

```
datetime.strptime("01 Jun, 2000", "%d %B, %Y")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [47], in <cell line: 1>()
----> 1 datetime.strptime("01 Jun, 2000", "%d %B, %Y")

File ~/anaconda3/lib/python3.9/_strptime.py:568, in _strptime_datetime(cls, data_string, format)
    565 def _strptime_datetime(cls, data_string, format="%a %b %d %H:%M:%S %Y"):
    566     """Return a class cls instance based on the input string and the
    567     format string."""
--> 568     tt, fraction, gmtoff_fraction = _strptime(data_string, format)
    569     tzname, gmtoff = tt[-2:]
    570     args = tt[:6] + (fraction,)

File ~/anaconda3/lib/python3.9/_strptime.py:349, in _strptime(data_string, format)
    347 found = format_regex.match(data_string)
    348 if not found:
--> 349     raise ValueError("time data %r does not match format %r" %
    350                      (data_string, format))
    351 if len(data_string) != found.end():
    352     raise ValueError("unconverted data remains: %s" %
    353                       data_string[found.end():])

ValueError: time data '01 Jun, 2000' does not match format '%d %B, %Y'
```

```
datetime.strptime("01 June, 2000", "%d %B, %Y")
```

```
datetime.datetime(2000, 6, 1, 0, 0)
```

```
datetime.strptime("01 June, 99", "%d %B, %Y")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [51], in <cell line: 1>()
----> 1 datetime.strptime("01 June, 99", "%d %B, %Y")

File ~/anaconda3/lib/python3.9/_strptime.py:568, in _strptime_datetime(cls, data_string, format)
    565 def _strptime_datetime(cls, data_string, format="%a %b %d %H:%M:%S %Y"):
    566     """Return a class cls instance based on the input string and the
    567     format string."""
--> 568     tt, fraction, gmtoff_fraction = _strptime(data_string, format)
    569     tzname, gmtoff = tt[-2:]
    570     args = tt[:6] + (fraction,)

File ~/anaconda3/lib/python3.9/_strptime.py:349, in _strptime(data_string, format)
    347 found = format_regex.match(data_string)
    348 if not found:
--> 349     raise ValueError("time data %r does not match format %r" %
    350                      (data_string, format))
    351 if len(data_string) != found.end():
    352     raise ValueError("unconverted data remains: %s" %
    353                       data_string[found.end():])

ValueError: time data '01 June, 99' does not match format '%d %B, %Y'
```

```
datetime.strptime("01 June, 99", "%d %B, %y")
```

```
datetime.datetime(1999, 6, 1, 0, 0)
```

Python strptime()

The `strptime()` method creates a datetime object from the given string.

Python strftime()

The `strftime()` method returns a string representing date and time using date, time or datetime object.

```
egu_start_day=datetime.strptime("14 Apr, 2024", "%d %b, %Y")
```

```
year_of_egu=egu_start_day.strftime("%Y")
print(year_of_egu)
```

```
2024
```

```
glob.glob('/                    Any path                    /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('        Path to data        /TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/ Any path /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

Selecting/SLICING file (variables)

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

```
glob.glob('/ Any path /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

Selecting/SLICING file (variables)

100% tqdm 22/22 [02:23<00:0 s/it]

```
KeyError: 'longitude is not a valid dimension or coordinate'
```

What if we give ‘longitude’ instead of ‘lon’

```
#A look at the data
#Note: We can use terminal comands in Jupyter notebook by using "!" sign
!ncdump -h Path to data/TRMM_daily_0.25x0.25_1998.nc
```

```
netcdf TRMM_daily_0.25x0.25_1998 {
dimensions:
        time = UNLIMITED ; // (365 currently)
        lon = 1440 ;
        lat = 400 ;
variables:
        double time(time) ;
                time:standard_name = "time" ;
                time:units = "hours since 1-1-1 00:00:00" ;
                time:calendar = "standard" ;
                time:axis = "T" ;
        double lon(lon) ;
                lon:standard_name = "longitude" ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
        double lat(lat) ;
                lat:standard_name = "latitude" ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
        float r(time, lat, lon) ;
                r:long_name = "Daily accumulated precipitation (combined microwave-IR) estimate with gauge calibration over lan
d  [mm]" ;
                r:_FillValue = -9999.9f ;
                r:missing_value = -9999.9f ;
```

Its TRMM3b42 version 7 0.25x0.25 degree gridded data listed yearly

```
glob.glob('/ Any path /*')
```

Same as doing “ls”

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2


seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('Path to data/TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                          lon=slice(int(west_lon),int(east_lon)),
                                          lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

To see progress bar (optional)

Reading file name

Reading file (variables)

Selecting/SLICING file (variables)

Computing mean for the selected data

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

What is appending in code?

append() is a method that adds (an) additional element(s) to the end of the selected parent element.

```
glob.glob('/                Any path                /*')
```

Same as doing "ls"

```
# 1. Read data year by year
# 2. Select (slice) the duration (time) and region (longitude and latitude) of interest
# 3. Compute average of the selected data in step 2

seasonal_mean=[]

for i in tqdm(range(start_year, end_year_idx)): #looping over files
    input_file=glob.glob('          Path to data          /TRMM_daily_0.25x0.25_' + str(i) + '.nc');
    input_variable=xr.open_dataset(input_file[0]);

    season_start_day = datetime.strptime("01 Jun, "+str(i)+"", "%d %b, %Y");
    season_end_day   = datetime.strptime("30 Sep, "+str(i)+"", "%d %b, %Y");

    seasonal_mean_yearwise=input_variable.sel(time=slice(season_start_day,season_end_day),
                                              lon=slice(int(west_lon),int(east_lon)),
                                              lat=slice(int(south_lat),int(north_lat))).mean(dim=['time','lon','lat']);
    #Check area weighting
    seasonal_mean.append(seasonal_mean_yearwise);
```

100% tqdm 22/22 [02:23<00:00, 7.88s/it]

To see progress bar (optional)

Reading file name

Reading file (variables)

Selecting/SLICING file (variables)

Computing mean for the selected data

Appending computed means

```
1 #At this point "seasonal_mean" is not a time series
2 seasonal_mean
```

```
[<xarray.Dataset>
 Dimensions:  ()
 Data variables:
     r        float32 7.953,
 <xarray.Dataset>
 Dimensions:  ()
 Data variables:
     r        float32 8.526,
 <xarray.Dataset>
 Dimensions:  ()
 Data variables:
     r        float32 7.145,
 <xarray.Dataset>
 Dimensions:  ()
 Data variables:
     r        float32 7.112,
 <xarray.Dataset>
 Dimensions:  ()
 Data variables:
     r        float32 6.147,
 <xarray.Dataset>
```

Just appended

xarray.concat

Concatenate xarray objects along a new or existing dimension.

```python
# 4. Store averagred data in one time series

# Concatenate xarray objects along a new or existing dimension.

seasonal_mean_timeseries=xr.concat(seasonal_mean,dim='year')
seasonal_mean_timeseries["year"]= np.arange(start_year,end_year_idx,1)
```

Assigning dimension and dimension name

```python
seasonal_mean_timeseries
```

xarray.Dataset

▸ Dimensions: (**year**: 22)

▾ Coordinates:

year	(year)	int64	1998 1999 2000 ... 2017 2018 2019

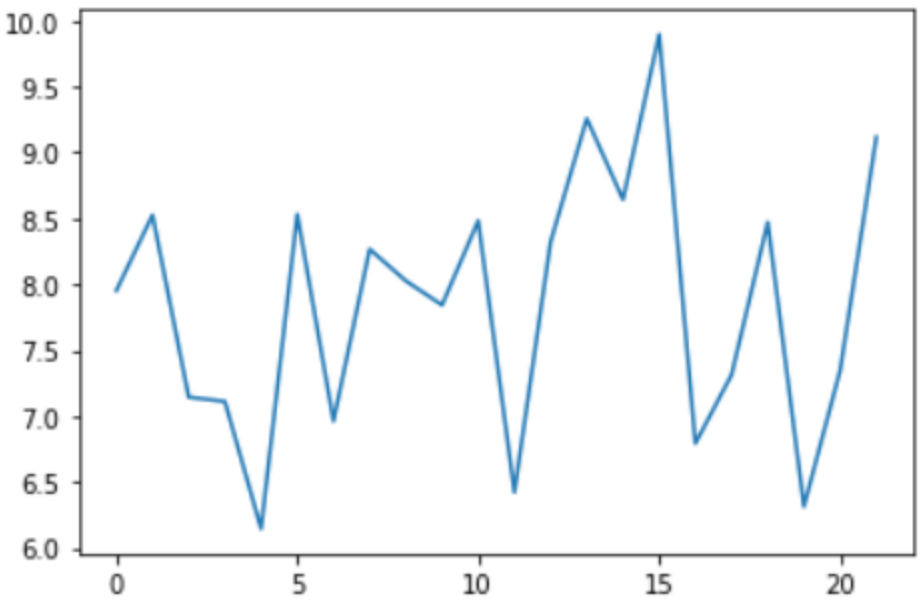
▾ Data variables:

r	(year)	float32	7.953 8.526 7.145 ... 7.347 9.118

▸ Attributes: (0)

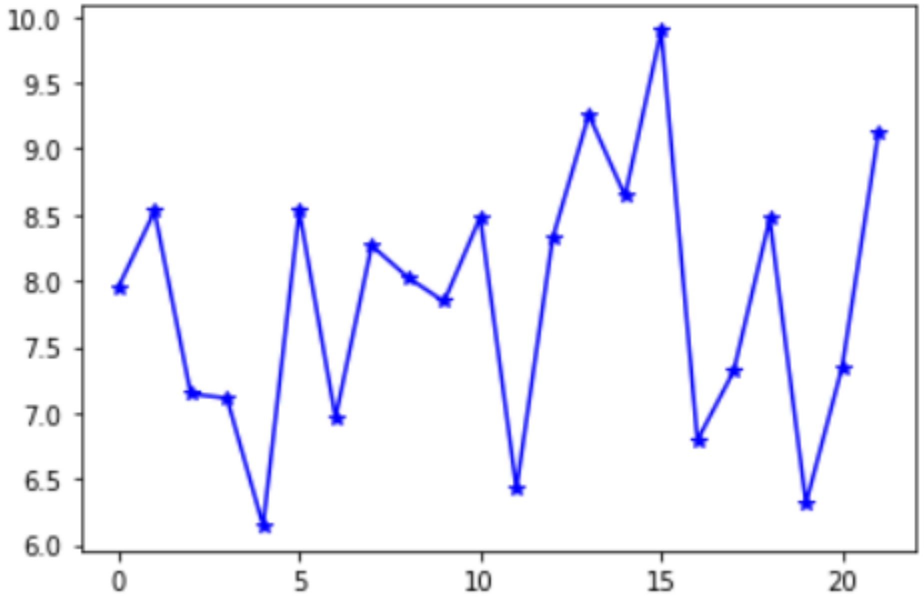
```python
# 5. plot the data

plt.plot(seasonal_mean_timeseries['r'])

```

```
[<matplotlib.lines.Line2D at 0x14e2fc694550>]
```



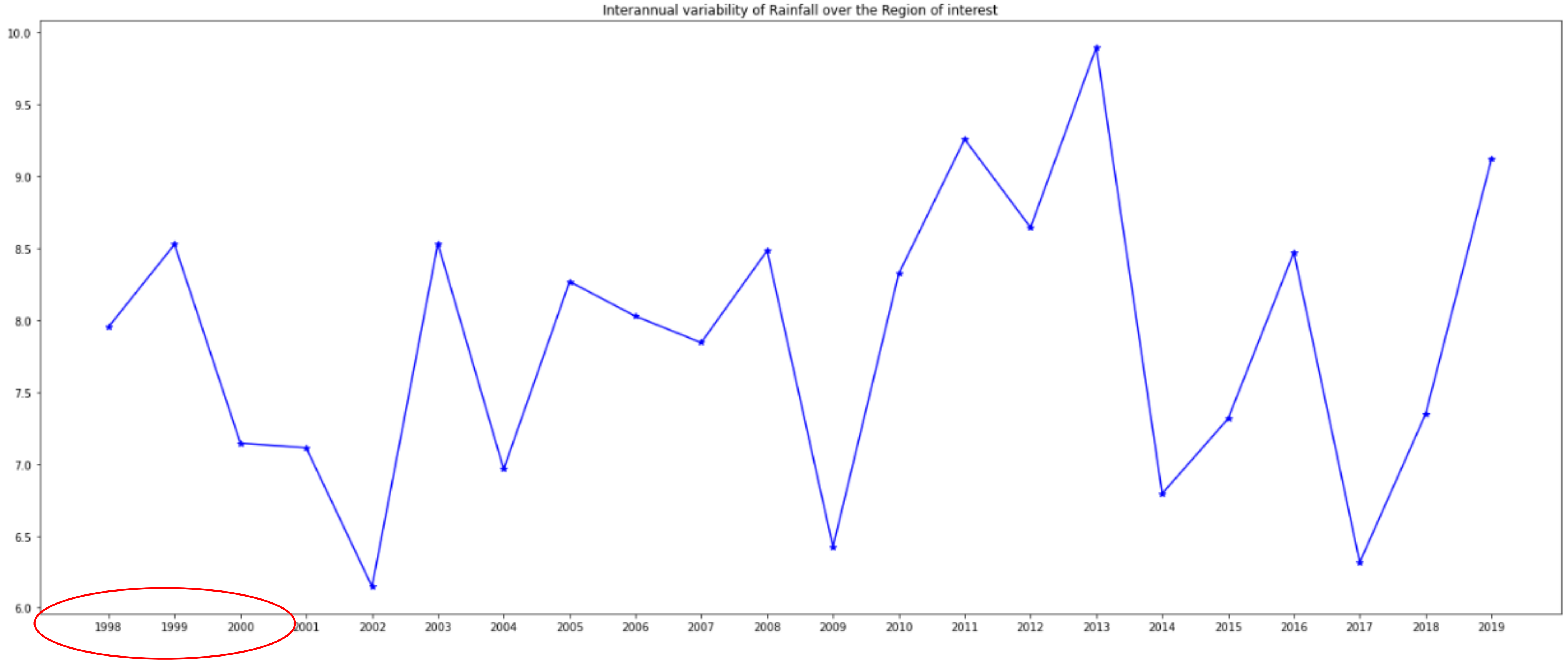
```python
# 6. Beautify the plot

plt.plot(seasonal_mean_timeseries['r'],'b-*')

```

```
[<matplotlib.lines.Line2D at 0x14e2fc404820>]
```



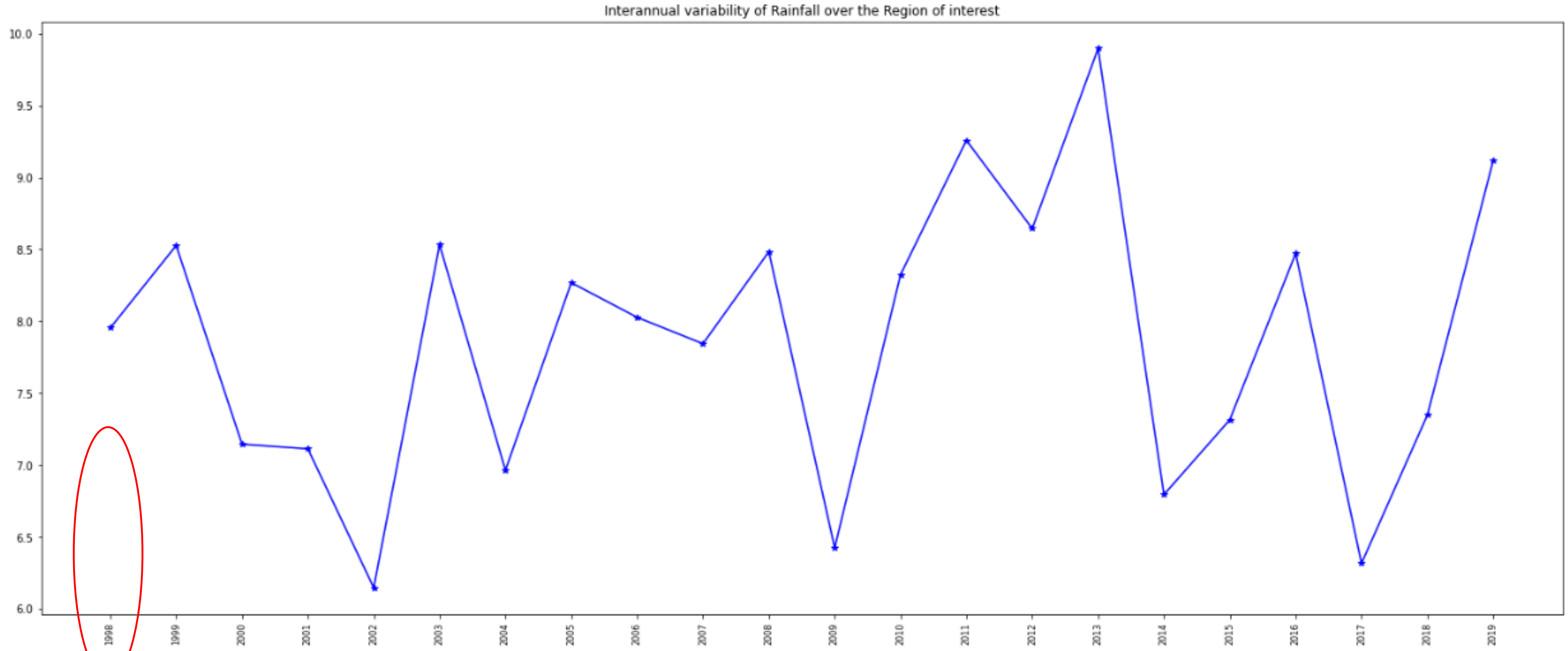
```
# 6. Beautify the plot
fig=plt.figure(figsize=(25,10))

plt.plot(seasonal_mean_timeseries['r'],'b-*')

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values);

plt.title("Interannual variability of Rainfall over the Region of interest")
```

```
Text(0.5, 1.0, 'Interannual variability of Rainfall over the Region of interest')
```



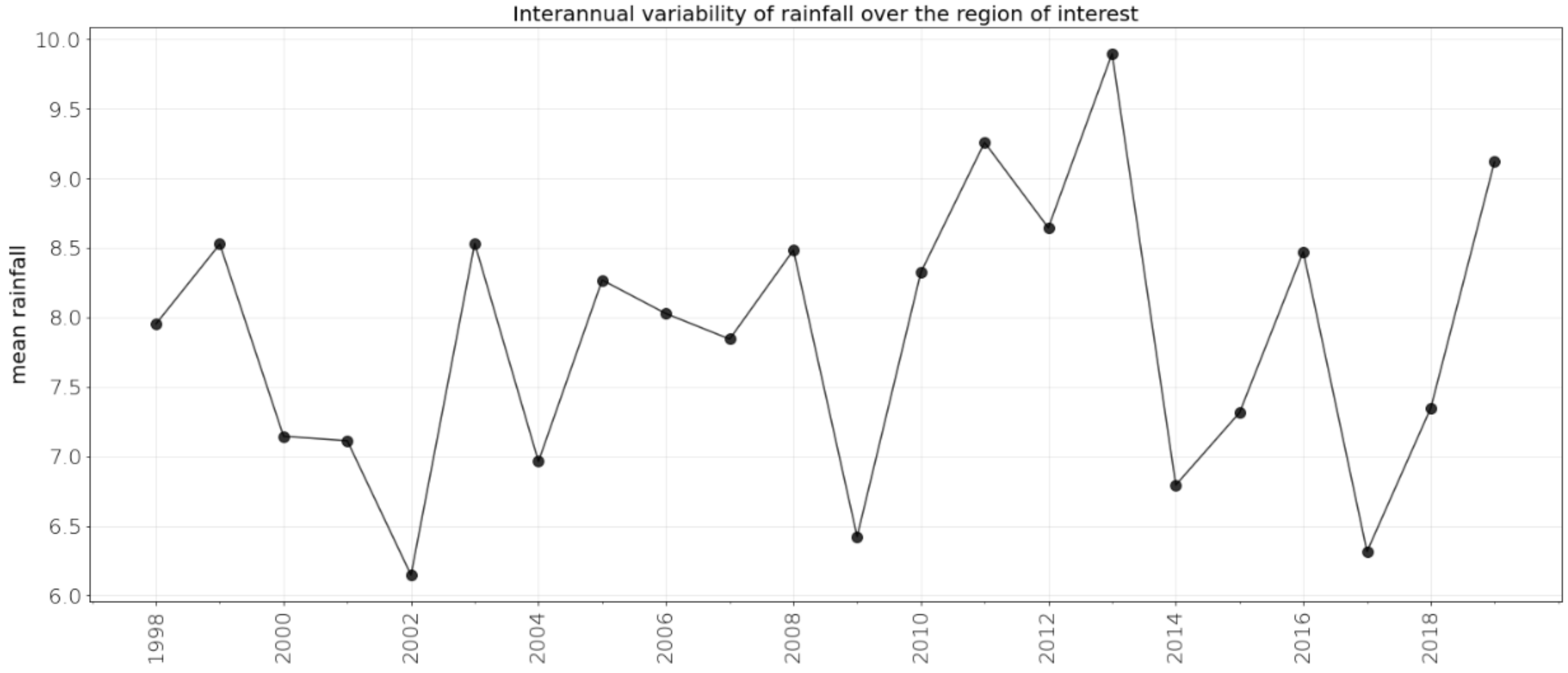
```python
# 6. Beautify the plot
fig=plt.figure(figsize=(25,10))

plt.plot(seasonal_mean_timeseries['r'],'b-*')

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values, rotation = 90,fontsize=8);

plt.title("Interannual variability of Rainfall over the Region of interest")
```

```
Text(0.5, 1.0, 'Interannual variability of Rainfall over the Region of interest')
```



```python
# 6. Beautify the plot
fig=plt.figure(figsize=(25,10))

plt.plot(seasonal_mean_timeseries['r'],color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='k')

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values, rotation = 90,fontsize=20, weight = 'light');
plt.yticks(fontsize=20, weight = 'light');
plt.grid(alpha=0.3)
plt.title("Interannual variability of rainfall over the region of interest",fontsize=20)
plt.ylabel("mean rainfall", fontsize=20)


from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
ax = plt.gca()
ax.set_xticks(ax.get_xticks()[::2])
ax.xaxis.set_minor_locator(MultipleLocator(1))
```

alternate x-axis tick labels



Task: 1 DONE

1. Read data year by year
2. Select (slice) the duration (time) and region (longitude and latitude) of interest
3. Compute average of the selected data in step 2
4. Store averagred data in one time series
5. plot the data
6. Beautify the plot

##

Task: 2

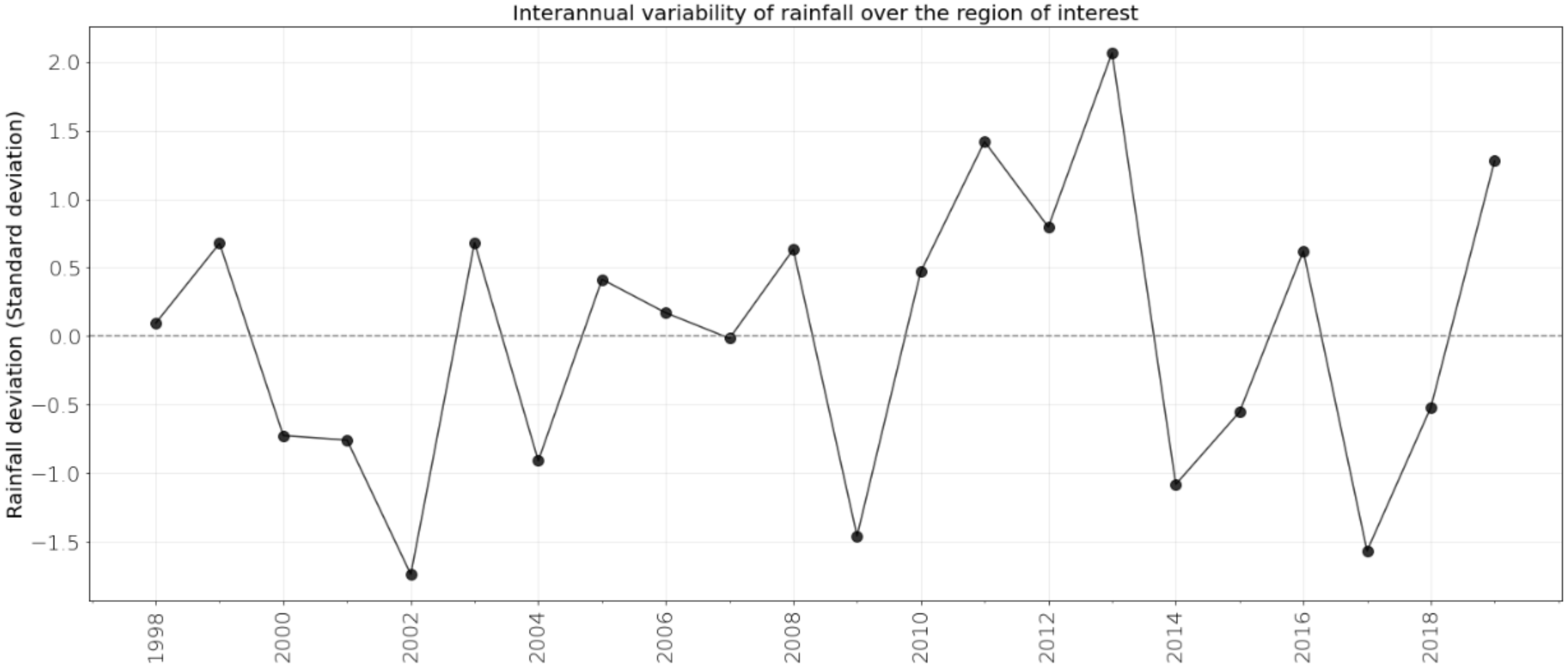
7. Compute SD of the time series
8. Identify excess/flood and deficit/drought years
9. Mark the flood and drought years in the data.

##

EASY

```python
#Compute long-term mean and deviation

mean_of_seasonal_mean_timeseries=seasonal_mean_timeseries.mean(dim='year')
std_of_seasonal_mean_timeseries=seasonal_mean_timeseries.std(dim='year')

#Compute deviation in unid of sigma
dev_of_seasonal_mean_timeseries=(seasonal_mean_timeseries-mean_of_seasonal_mean_timeseries)/std_of_seasonal_mean_timeseries
```

```python
#Plot interannual variability (in terms of deviation from the long-term mean)
fig=plt.figure(figsize=(25,10))

plt.plot(dev_of_seasonal_mean_timeseries['r'],color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='k')

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values, rotation = 90,fontsize=20, weight = 'light');
plt.yticks(fontsize=20, weight = 'light');
plt.grid(alpha=0.3)
plt.title("Interannual variability of rainfall over the region of interest",fontsize=20)
plt.ylabel("Rainfall deviation (Standard deviation)", fontsize=20)


##################################################
plt.axhline(y=0, color='grey', linestyle='--')  ;#
##################################################


from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
ax = plt.gca()
ax.set_xticks(ax.get_xticks()[::2])
ax.xaxis.set_minor_locator(MultipleLocator(1))
```



```python
#Plot interannual variability (in terms of deviation from the long-term mean)

fig=plt.figure(figsize=(25,10))

##################################################
#plt.plot(dev_of_seasonal_mean_timeseries['rf'],color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='k')
plt.plot(dev_of_seasonal_mean_timeseries['r'],color='k')
##################################################

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values, rotation = 90,fontsize=20, weight = 'light');
plt.yticks(fontsize=20, weight = 'light');
plt.grid(alpha=0.3)
plt.title("Interannual variability of rainfall over the region of interest",fontsize=20)
plt.ylabel("Rainfall deviation (Standard deviation)", fontsize=20)

plt.axhline(y=0, color='grey', linestyle='--')

from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
ax = plt.gca()
ax.set_xticks(ax.get_xticks()[::2])
ax.xaxis.set_minor_locator(MultipleLocator(1))



##################################################
########### Marking EXCESS & DEFICIT #############
##################################################
threshold_excess  =  1.0    ;# above 1 standard deviation
threshold_deficit = -1.0    ;# below -1 standard deviation

plt.axhline(y=0, color='k', linestyle=':')
plt.axhline(y=threshold_flood, color='gray', linestyle=':')
plt.axhline(y=threshold_drought, color='gray', linestyle=':')

rainfall_deviation_values = dev_of_seasonal_mean_timeseries['r']

greater_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val>threshold_excess]
smaller_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_deficit]
within_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_excess and val>threshold_deficit]

plt.plot(greater_than_threshold, rainfall_deviation_values[greater_than_threshold],
        linestyle='none', color='r', marker='o',markersize=10,alpha=0.8,markerfacecolor='r')

plt.plot(smaller_than_threshold, rainfall_deviation_values[smaller_than_threshold],
        linestyle='none', color='b', marker='o',markersize=10,alpha=0.8,markerfacecolor='b')

plt.plot(within_threshold, rainfall_deviation_values[within_threshold],
        linestyle='none', color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='none')

##################################################


plt.savefig('EGU_Figure_1.pdf', format='pdf', dpi=1200) ;#Publication ready
```

```python

##################################################
########### Marking EXCESS & DEFICIT #############
##################################################
threshold_excess  =  1.0    ;# above 1 standard deviation
threshold_deficit = -1.0    ;# below -1 standard deviation

plt.axhline(y=0, color='k', linestyle=':')
plt.axhline(y=threshold_flood, color='gray', linestyle=':')
plt.axhline(y=threshold_drought, color='gray', linestyle=':')

rainfall_deviation_values = dev_of_seasonal_mean_timeseries['r']

greater_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val>threshold_excess]
smaller_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_deficit]
within_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_excess and val>threshold_deficit]

plt.plot(greater_than_threshold, rainfall_deviation_values[greater_than_threshold],
        linestyle='none', color='r', marker='o',markersize=10,alpha=0.8,markerfacecolor='r')

plt.plot(smaller_than_threshold, rainfall_deviation_values[smaller_than_threshold],
        linestyle='none', color='b', marker='o',markersize=10,alpha=0.8,markerfacecolor='b')

plt.plot(within_threshold, rainfall_deviation_values[within_threshold],
        linestyle='none', color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='none')

##################################################
```

```python
greater_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val>threshold_excess]
```

Do nothing

Dictionary meaning:
mention one by one.

Condition

What Does the Enumerate Function in Python do?

The enumerate function in Python is a built-in function that allows programmers to loop over something and have an automatic counter.

https://www.geeksforgeeks.org/what-does-the-enumerate-function-in-python-do/

```python
sample_series = ([-10,5,1,-2,7])
```

```python
# printing the elements directly
for elements in enumerate(sample_series):
    print(elements)
```

```
(0, -10)
(1, 5)
(2, 1)
(3, -2)
(4, 7)
```

```python
my_list = [i for i, val in enumerate(sample_series) if val>0]
my_list
```

```
[1, 2, 4]
```

```python
my_list = [i for i, val in enumerate(sample_series) if val<0]
my_list
```

```
[0, 3]
```

```python
my_list = [i+5 for i, val in enumerate(sample_series) if val>0]
my_list
```

```
[6, 7, 9]
```

Add 5

```python
my_list = [i**3 for i, val in enumerate(sample_series) if val<0]
my_list
```

```
[0, 27]
```

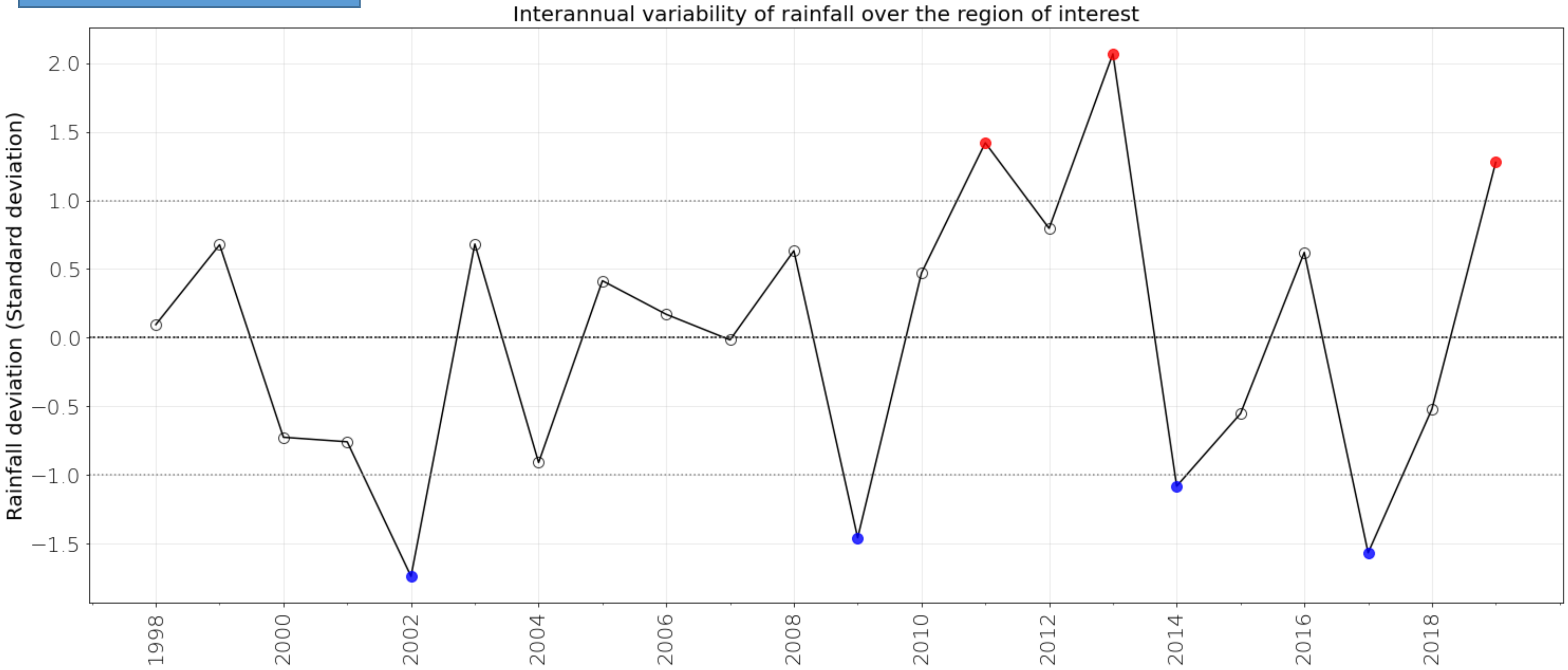
Raise to the power 3

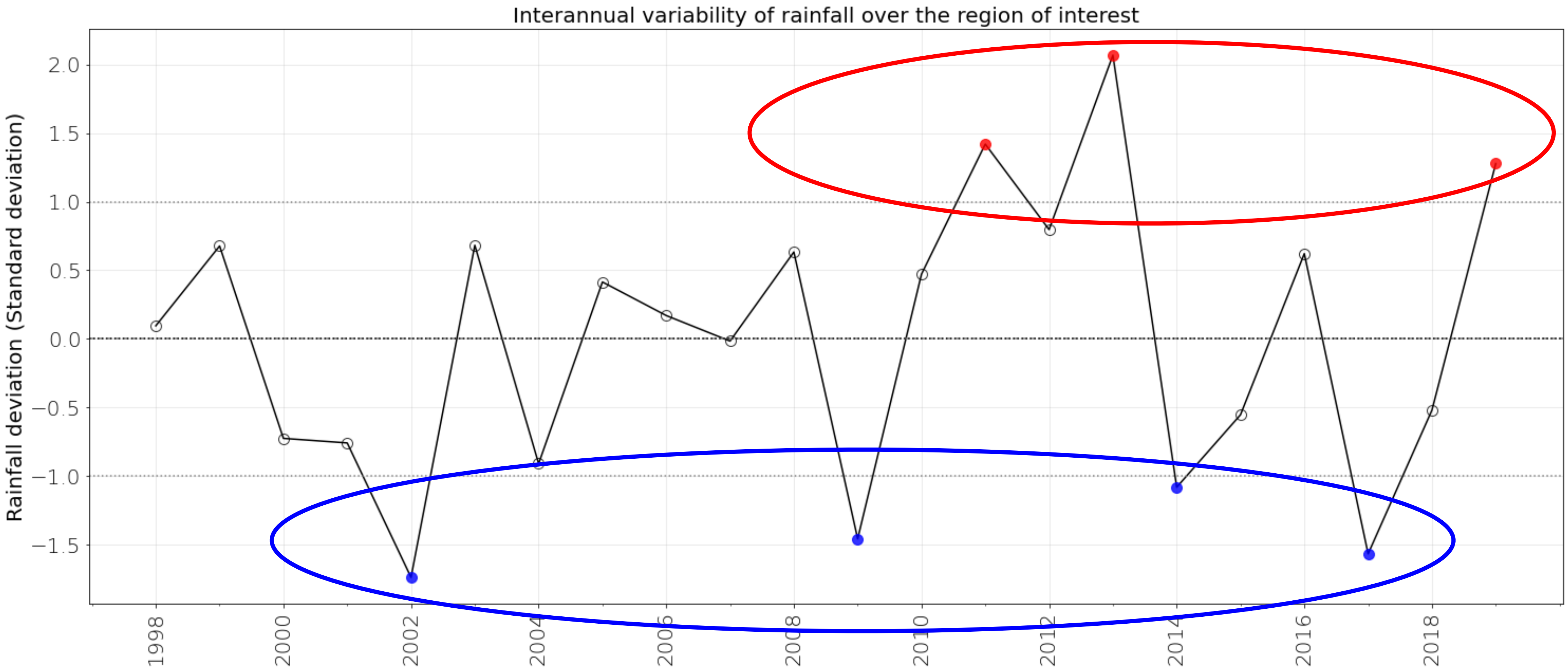
```python
#Plot interannual variability (in terms of deviation from the long-term mean)

fig=plt.figure(figsize=(25,10))

##################################################
#plt.plot(dev_of_seasonal_mean_timeseries['rf'],color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='k')
plt.plot(dev_of_seasonal_mean_timeseries['r'],color='k')
##################################################

default_x_ticks = range(len(seasonal_mean_timeseries['year'].values));
plt.xticks(default_x_ticks, seasonal_mean_timeseries['year'].values, rotation = 90,fontsize=20, weight = 'light');
plt.yticks(fontsize=20, weight = 'light');
plt.grid(alpha=0.3)
plt.title("Interannual variability of rainfall over the region of interest",fontsize=20)
plt.ylabel("Rainfall deviation (Standard deviation)", fontsize=20)

plt.axhline(y=0, color='grey', linestyle='--')

from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
ax = plt.gca()
ax.set_xticks(ax.get_xticks()[::2])
ax.xaxis.set_minor_locator(MultipleLocator(1))



##################################################
########### Marking EXCESS & DEFICIT #############
##################################################
threshold_excess  =  1.0    ;# above 1 standard deviation
threshold_deficit = -1.0    ;# below -1 standard deviation

plt.axhline(y=0, color='k', linestyle=':')
plt.axhline(y=threshold_flood, color='gray', linestyle=':')
plt.axhline(y=threshold_drought, color='gray', linestyle=':')

rainfall_deviation_values = dev_of_seasonal_mean_timeseries['r']

greater_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val>threshold_excess]
smaller_than_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_deficit]
within_threshold = [i for i, val in enumerate(rainfall_deviation_values) if val<threshold_excess and val>threshold_deficit]

plt.plot(greater_than_threshold, rainfall_deviation_values[greater_than_threshold],
        linestyle='none', color='r', marker='o',markersize=10,alpha=0.8,markerfacecolor='r')

plt.plot(smaller_than_threshold, rainfall_deviation_values[smaller_than_threshold],
        linestyle='none', color='b', marker='o',markersize=10,alpha=0.8,markerfacecolor='b')

plt.plot(within_threshold, rainfall_deviation_values[within_threshold],
        linestyle='none', color='k', marker='o',markersize=10,alpha=0.8,markerfacecolor='none')

##################################################


plt.savefig('EGU_Figure_1.pdf', format='pdf', dpi=1200) ;#Publication ready

```

Identify Excess, Deficit, etc.

Plot only marker

Publication ready plot
(dpi=1200)




Interannual variability of rainfall over the region of interest
Rainfall deviation (Standard deviation)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018

```
#Task 3
```

```

##The COOLEST part

excess_years=dev_of_seasonal_mean_timeseries['year'].values[greater_than_threshold[:]]
deficit_years=dev_of_seasonal_mean_timeseries['year'].values[smaller_than_threshold[:]]

# ##########################################################################################

# Suppose I want to save only those years during  1971-1990

# yrstart=np.where((dev_of_seasonal_mean_timeseries.year.values==1971).squeeze() > 0)[0][0]
# yrend=np.where((dev_of_seasonal_mean_timeseries.year.values==1990).squeeze() > 0)[0][0]

# yrstart_idx=np.where((greater_than_threshold[:]>yrstart)> 0)[0][0]
# yrend_idx=np.where((greater_than_threshold[:]>yrend)> 0)[0][0]

# excess_years=dev_of_seasonal_mean_timeseries['year'].values[greater_than_threshold[yrstart_idx:yrend_idx]]
# deficit_years=dev_of_seasonal_mean_timeseries['year'].values[smaller_than_threshold[yrstart_idx:yrend_idx]]
# ##########################################################################################

print(excess_years)
print(deficit_years)
```

```
[2011 2013 2019]
[2002 2009 2014 2017]
```

```
%store excess_years
%store deficit_years
```

```
Stored 'excess_years' (ndarray)
Stored 'deficit_years' (ndarray)
```

Stored where?
(Note: My current file name is Python_Course_EGU_File01)

jupyter Python_Course_EGU_File02 Last Checkpoint: 03/27/2024 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

Run Code

In [1]:

```
#show how we can use stored variables
```

In [2]:

```
import numpy as np
import scipy as sc
import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import xarray as xr
import glob
from tqdm.notebook import tqdm
```

In [3]:

```
###Recall

%store -r excess_years
%store -r deficit_years
```

In [4]:

```
print(excess_years)
print(deficit_years)
```

```
[2011 2013 2019]
[2002 2009 2014 2017]
```

In [5]:

```
print("Number of EXCESS Years = ",len(excess_years))
print("Number of DEFICIT Years = ",len(deficit_years))
```

```
Number of EXCESS Years =  3
Number of DEFICIT Years =  4
```

jupyter Python_Course_EGU_File02 Last Checkpoint: 03/27/2024 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:

```
#show how we can use stored variables
```

In [2]:

```
import numpy as np
import scipy as sc
import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import xarray as xr
import glob
from tqdm.notebook import tqdm
```

In [3]:

```
###Recall

%store -r excess_years
%store -r deficit_years
```

Find it yourself: how to delete stored variable(s)

In [4]:

```
print(excess_years)
print(deficit_years)
```

```
[2011 2013 2019]
[2002 2009 2014 2017]
```

In [5]:

```
print("Number of EXCESS Years = ",len(excess_years))
print("Number of DEFICIT Years = ",len(deficit_years))
```

```
Number of EXCESS Years =  3
Number of DEFICIT Years =  4
```

```
print("Excess")
excess_clim_temporary=[]
for i in excess_years:
    print(i)
    excess_file=xr.open_dataset('/ Path to data /Bidyut_Goswami/OBS/HadSST/HadISST-'+str(i)+'.nc')
    excess_clim_temporary.append(excess_file)

excess_clim=xr.concat(excess_clim_temporary,dim='time')


print("Deficit")
deficit_clim_temporary=[]
for i in deficit_years:
    print(i)
    deficit_file=xr.open_dataset('/ Path to data /Bidyut_Goswami/OBS/HadSST/HadISST-'+str(i)+'.nc')
    deficit_clim_temporary.append(deficit_file)

deficit_clim=xr.concat(deficit_clim_temporary,dim='time')
```

- Sea surface temperature data
- Reading
- Concatenating …

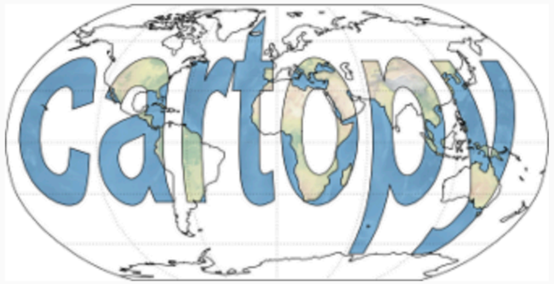
```
Excess
2011
2013
2019
Deficit
2002
2009
2014
2017
```

```python
#Computing mean (Composite) of SST
sst_excess  = excess_clim.mean(dim='time')
sst_deficit = deficit_clim.mean(dim='time')

#Compute difference between Deficit and Excess
sst_deficit_minus_sst_excess=sst_deficit-sst_excess

#Select data from 50S to 50N
LatBound=50
sst=sst_deficit_minus_sst_excess.sel(latitude=slice(LatBound,-LatBound))
```

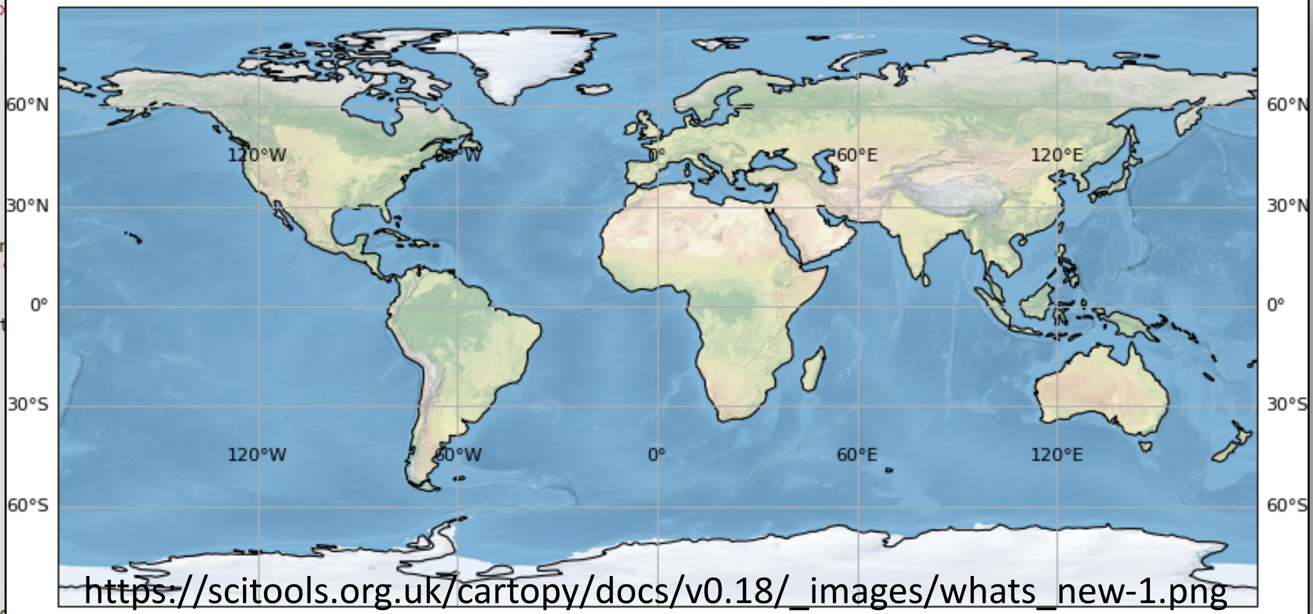
Remember me!!!
I am “slice”

```python
#Plot
lon=sst.variables['longitude'][:]
lat=sst.variables['latitude'][:]

lon2D, lat2D = np.meshgrid(lon, lat)
cl1=-1
cl2=-cl1
cnum=21
clevs=np.linspace(cl1,cl2,cnum)

import matplotlib
import cartopy.feature as cf


fig=plt.figure(figsize=(15,6))
axl=plt.axes(projection=ccrs.PlateCarree(central_longitude=180), facecolor='none')
axl.contourf(lon2D-180, lat2D, sst['sst'], clevs, extend='both', cmap='RdBu_r',
             projection=ccrs.PlateCarree(central_longitude=180))
axl.coastlines()
axl.gridlines(draw_labels=True, alpha=0.5)

axl.set_title('SST (Summer) Deficit(Drought)-Excess(Flood) ($\degree$C)', fontsize=20)
#######################################
lowbound=cl1
highbound=cl2
#intervel_num=15
#For plotting shared colorbar
postop1= axl.get_position()
posbot1= axl.get_position()

norm = matplotlib.colors.Normalize(vmin=lowbound, vmax=highbound,clip=True)
sm = matplotlib.cm.ScalarMappable(cmap='RdBu_r', norm=norm)
sm.set_array([])
cbar_ax = fig.add_axes([postop1.y1+0.15, posbot1.y0, posbot1.width/50, postop1.y1-posbot1.y0])
cbar_ax.tick_params(labelsize=14, width=0.2)
plt.colorbar(sm, cax=cbar_ax)

#########################################
#Region boundaries
west_lon = 75.0
east_lon = 84.0
south_lat= 18.0
north_lat= 28.0

from shapely.geometry.polygon import LinearRing
lons = [west_lon,  west_lon,  east_lon,  east_lon]
lats = [south_lat, north_lat, north_lat, south_lat]
ring = LinearRing(list(zip(lons, lats)))
axl.add_geometries([ring], ccrs.PlateCarree(), facecolor='none', edgecolor='black',linewidth=2)

plt.savefig('EGU_Figure_2.pdf', format='pdf', dpi=1200) ;#Publication ready
```

```
#Plot
lon=sst.variables['longitude'][:]
lat=sst.variables['latitude'][:]

lon2D, lat2D = np.meshgrid(lon, lat)
cl1=-1
cl2=-cl1
cnum=21
clevs=np.linspace(cl1,cl2,cnum)

import matplotlib
import cartopy.feature as cf


fig=plt.figure(figsize=(15,6))
axl=plt.axes(projection=ccrs.PlateCarree(centra
axl.contourf(lon2D-180, lat2D, sst['sst'], clev
             projection=ccrs.PlateCarree(centra
axl.coastlines()
axl.gridlines(draw_labels=True, alpha=0.5)

axl.set_title('SST (Summer) Deficit(Drought)-Ex
#######################################
lowbound=cl1
highbound=cl2
#intervel_num=15
#For plotting shared colorbar
postop1= axl.get_position()
posbot1= axl.get_position()

norm = matplotlib.colors.Normalize(vmin=lowbour
sm = matplotlib.cm.ScalarMappable(cmap='RdBu_r'
sm.set_array([])
cbar_ax = fig.add_axes([postop1.y1+0.15, posbot
cbar_ax.tick_params(labelsize=14, width=0.2)
plt.colorbar(sm, cax=cbar_ax)

###########################################
#Region boundaries
west_lon = 75.0
east_lon = 84.0
south_lat= 18.0
north_lat= 28.0

from shapely.geometry.polygon import LinearRing
lons = [west_lon,  west_lon,  east_lon,  east_lon]
lats = [south_lat, north_lat, north_lat, south_lat]
ring = LinearRing(list(zip(lons, lats)))
axl.add_geometries([ring], ccrs.PlateCarree(), facecolor='none', edgecolor='black',linewidth=2)

plt.savefig('EGU_Figure_2.pdf', format='pdf', dpi=1200) ;#Publication ready
```



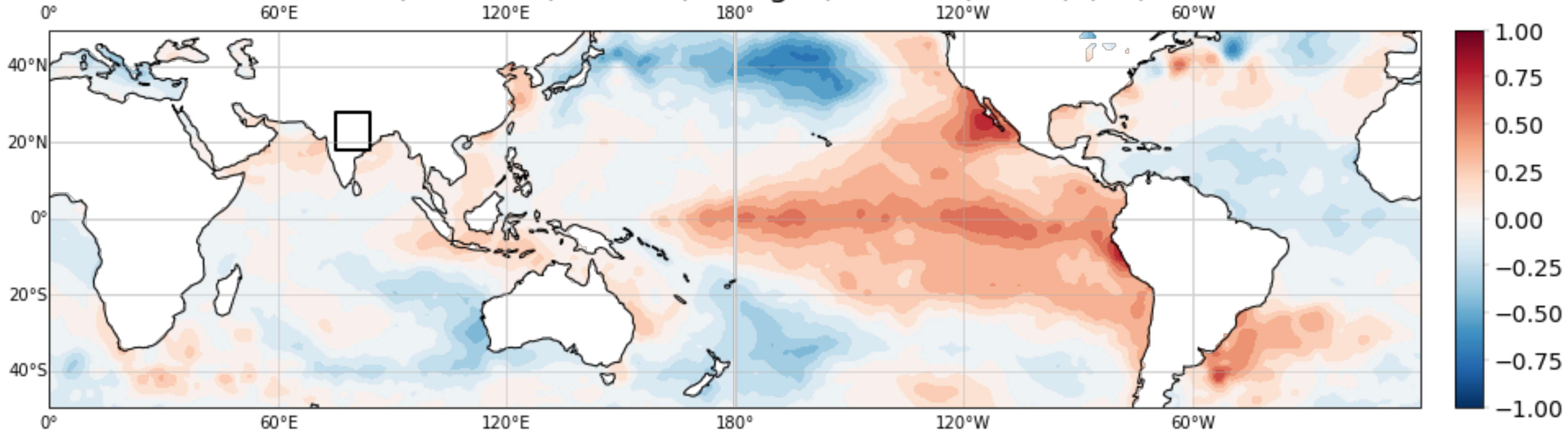
Cartopy is a Python package designed to make drawing maps for data analysis and visualisation easy.

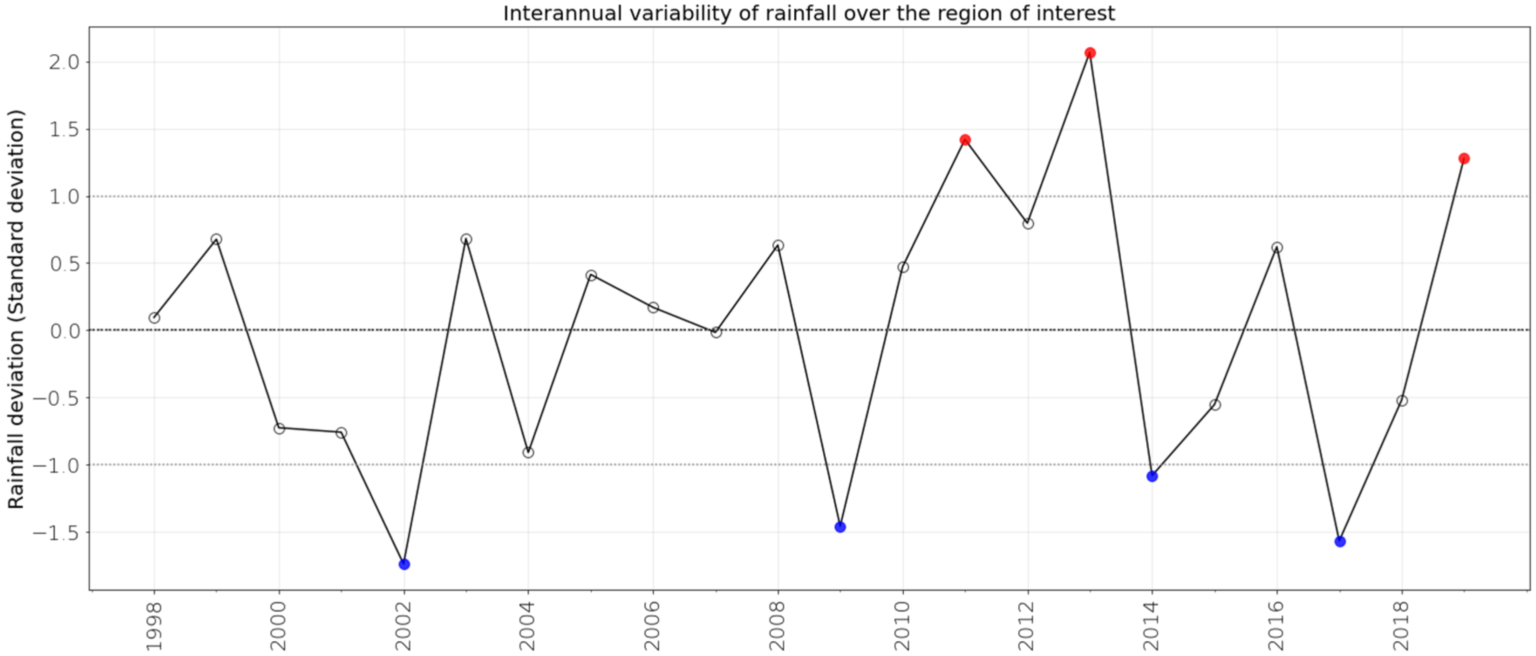


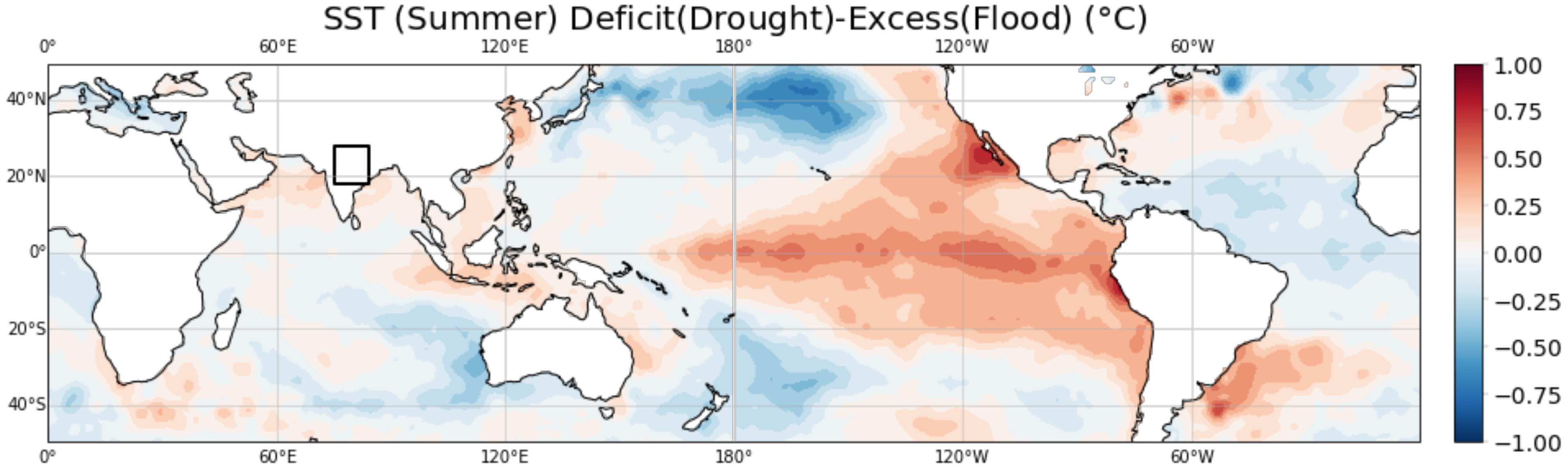
https://scitools.org.uk/cartopy/docs/v0.18/_images/whats_new-1.png

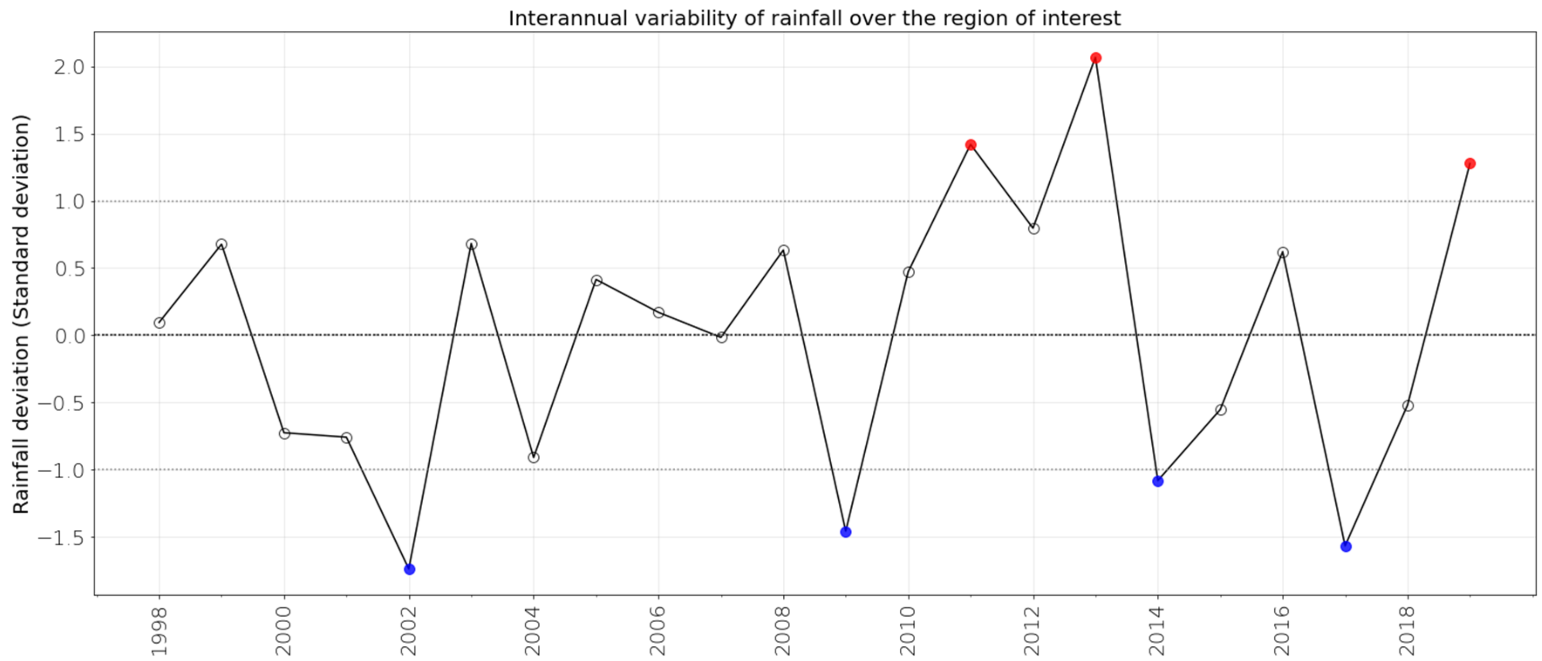
```python
#Plot
lon=sst.variables['longitude'][:]
lat=sst.variables['latitude'][:]

lon2D, lat2D = np.meshgrid(lon, lat)
cl1=-1
cl2=-cl1
cnum=21
clevs=np.linspace(cl1,cl2,cnum)

import matplotlib
import cartopy.feature as cf


fig=plt.figure(figsize=(15,6))
axl=plt.axes(projection=ccrs.PlateCarree(central_longitude=180), facecolor='none')
axl.contourf(lon2D-180, lat2D, sst['sst'], clevs, extend='both', cmap='RdBu_r',
             projection=ccrs.PlateCarree(central_longitude=180))
axl.coastlines()
axl.gridlines(draw_labels=True, alpha=0.5)

axl.set_title('SST (Summer) Deficit(Drought)-Excess(Flood) ($\degree$C)', fontsize=20)
#######################################
lowbound=cl1
highbound=cl2
#interval_num=15
#For plotting shared colorbar
postop1= axl.get_position()
posbot1= axl.get_position()

norm = matplotlib.colors.Normalize(vmin=lowbound, vmax=highbound,clip=True)
sm = matplotlib.cm.ScalarMappable(cmap='RdBu_r', norm=norm)
sm.set_array([])
cbar_ax = fig.add_axes([postop1.y1+0.15, posbot1.y0, posbot1.width/50, postop1.y1-posbot1.y0])
cbar_ax.tick_params(labelsize=14, width=0.2)
plt.colorbar(sm, cax=cbar_ax)

#########################################
#Region boundaries
west_lon = 75.0
east_lon = 84.0
south_lat= 18.0
north_lat= 28.0

from shapely.geometry.polygon import LinearRing
lons = [west_lon,  west_lon,  east_lon,  east_lon]
lats = [south_lat, north_lat, north_lat, south_lat]
ring = LinearRing(list(zip(lons, lats)))
axl.add_geometries([ring], ccrs.PlateCarree(), facecolor='none', edgecolor='black',linewidth=2)

plt.savefig('EGU_Figure_2.pdf', format='pdf', dpi=1200) ;#Publication ready
```

SST (Summer) Deficit(Drought)-Excess(Flood) (°C)

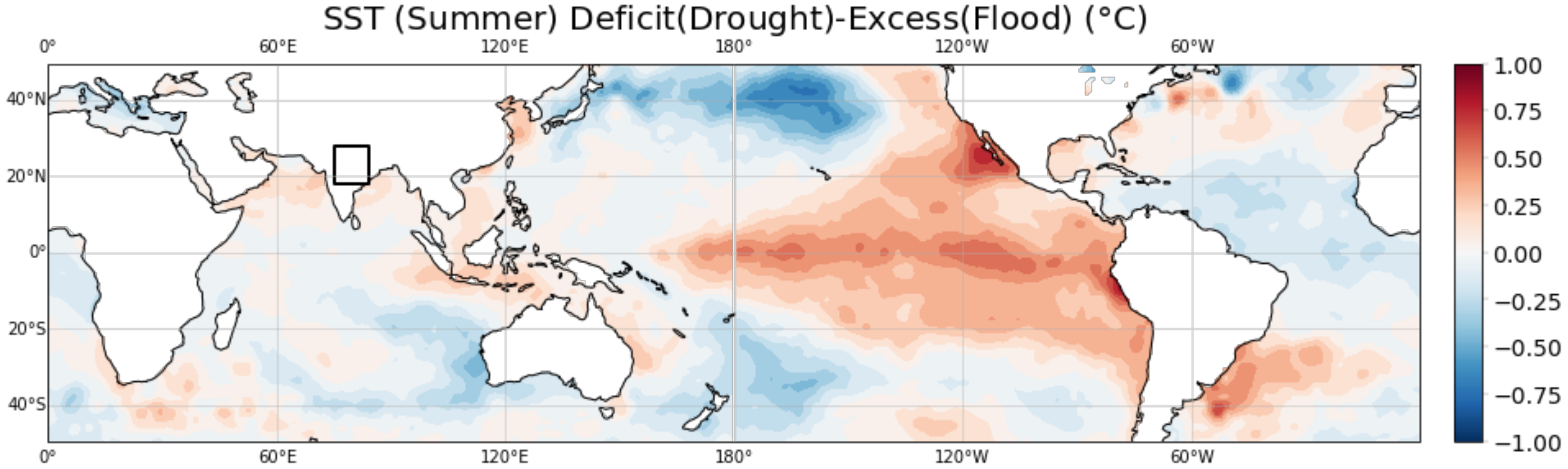



Interannual variability of rainfall over the region of interest
Rainfall deviation (Standard deviation)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018


SST (Summer) Deficit(Drought)-Excess(Flood) (°C)
0°
60°E
120°E
180°
120°W
60°W
40°N
20°N
0°
20°S
40°S
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



Scientific interpretation of our analysis: Indian monsoon drought is commonly associated with an El Nino like SST pattern



Thank you