HOMEWORK 7

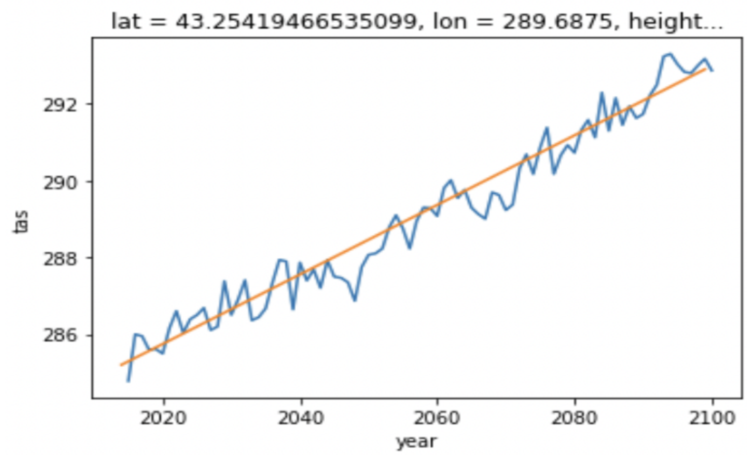
1. Climate data analysis

Step 1: **Chosen urban region: Boston; Chosen Rural region: Amherst**

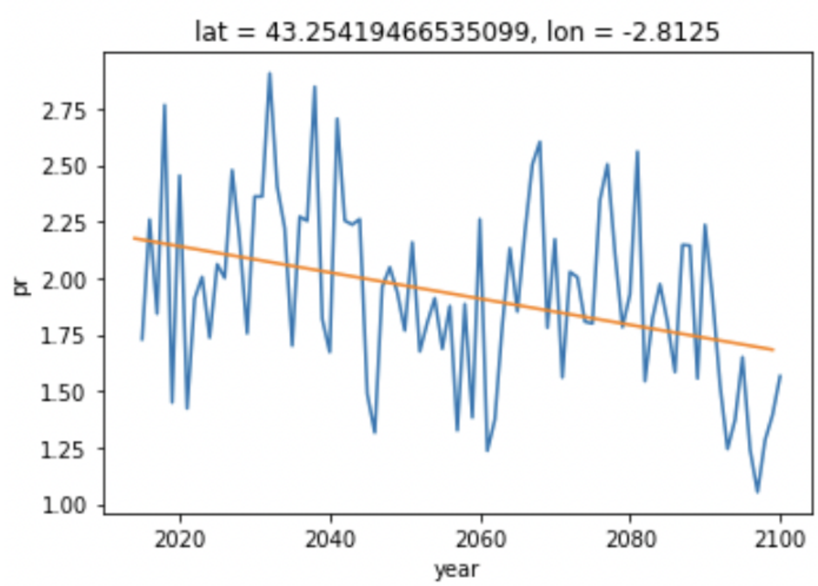
Step 2:

For Urban region: Boston

Future Projections for Surface Air Temperature (tas)

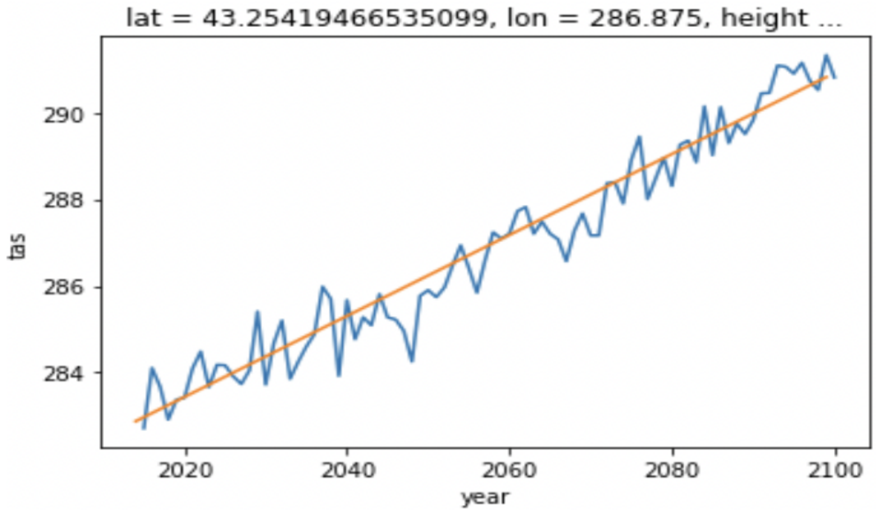


Future Projections for Precipitation(pr)

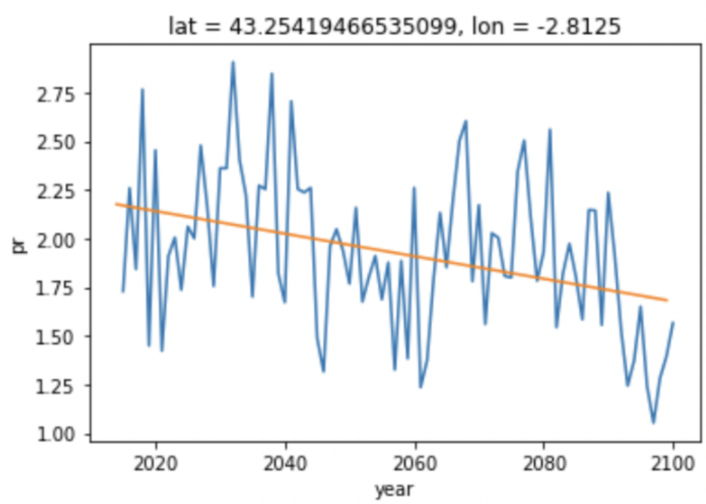


For Rural region: Amherst

Future Projections for Surface Air Temperature (tas)



Future Projections for Precipitation(pr)

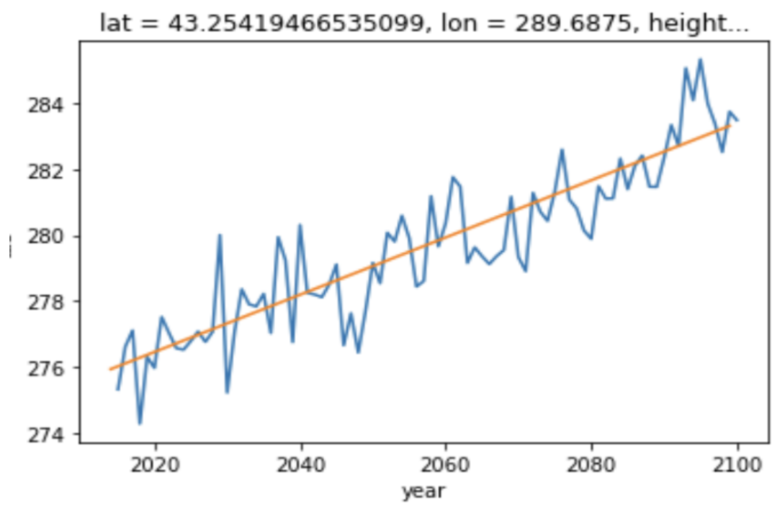


Step 3:

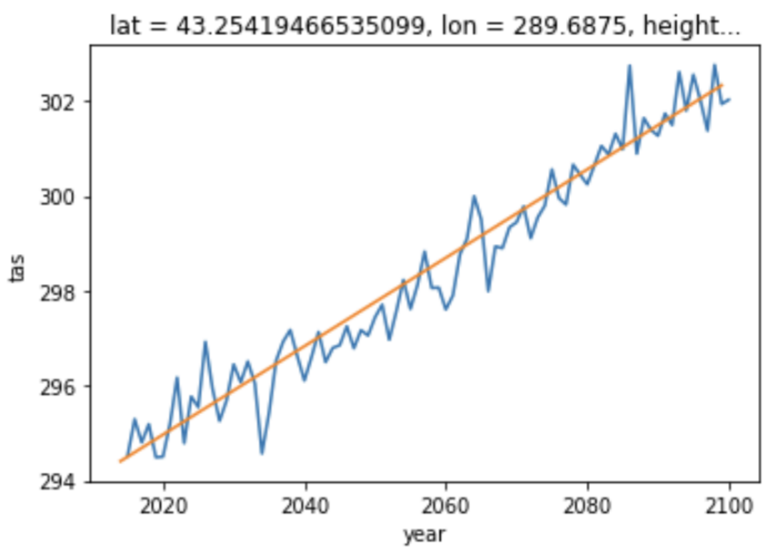
Region: Boston

DJF and JJA for Surface air Temperature (tas):

1.DJF tas

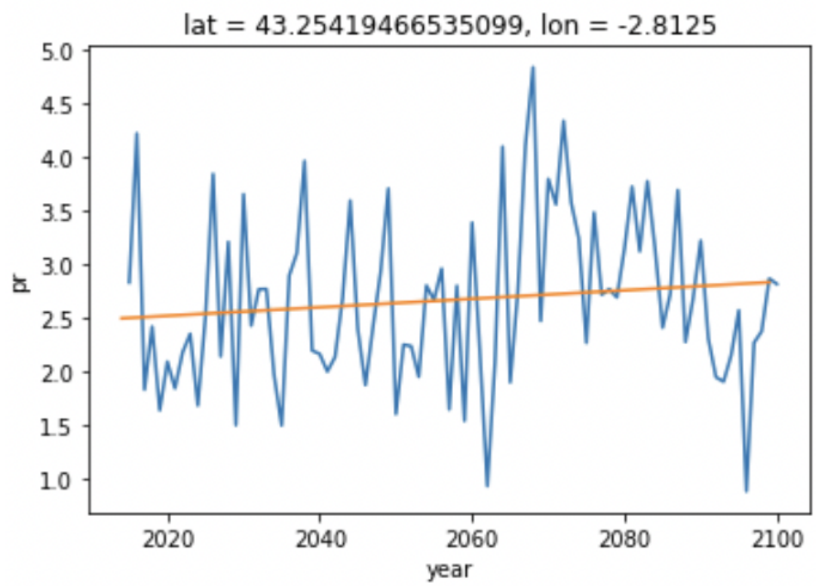


2. JJA tas

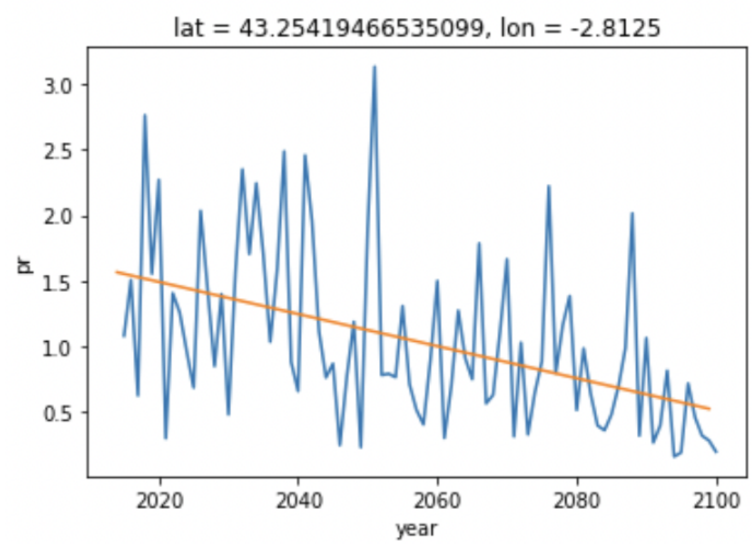


DJF and JJA for Precipitation(pr):

1. **DJF Precipitation**



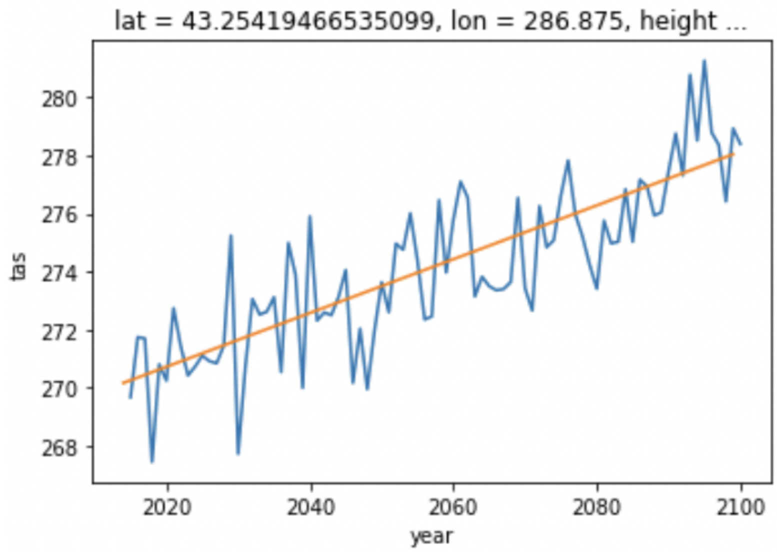
2. **JJA Precipitation**



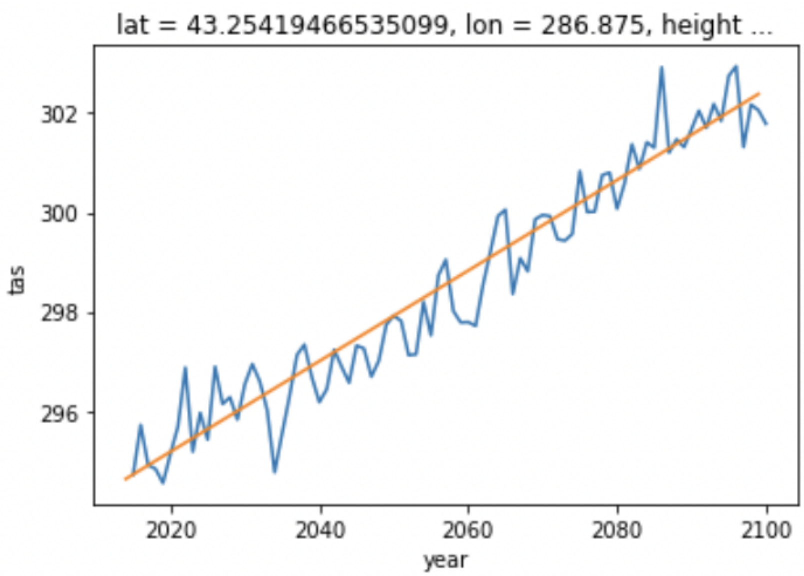
Region: Amherst

DJF and JJA for Surface air Temperature (tas):

1.DJF tas

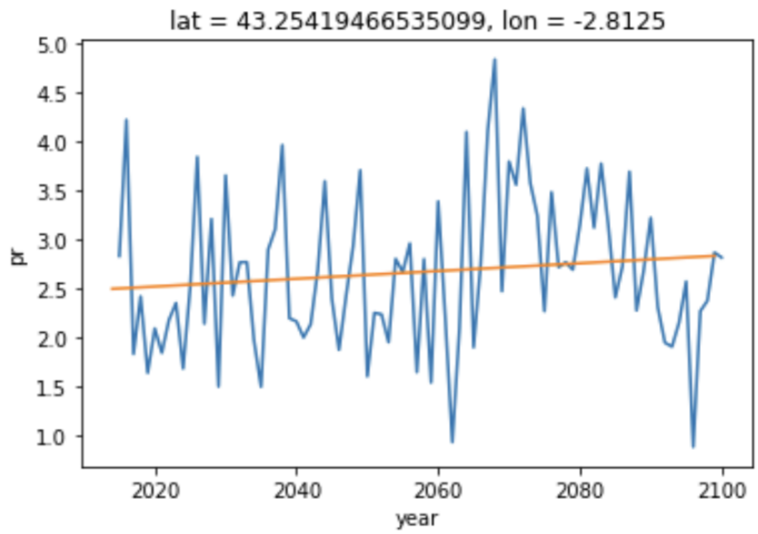


2. JJA tas

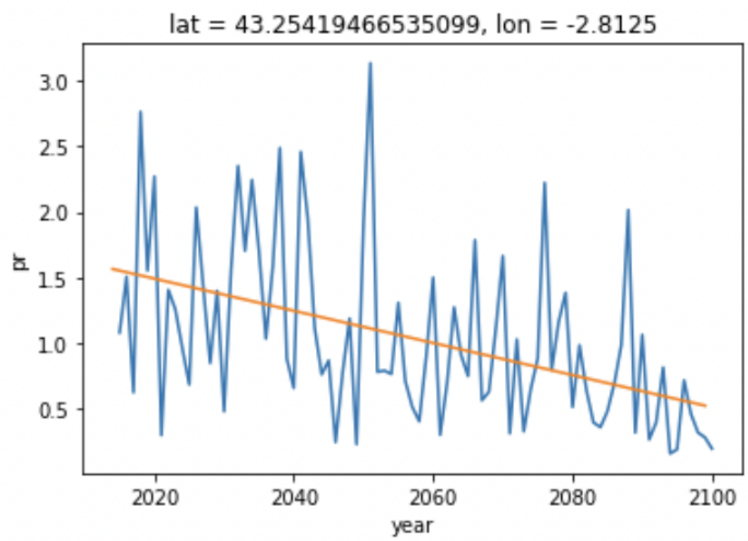


DJF and JJA for Precipitation(pr):

3. DJF Precipitation



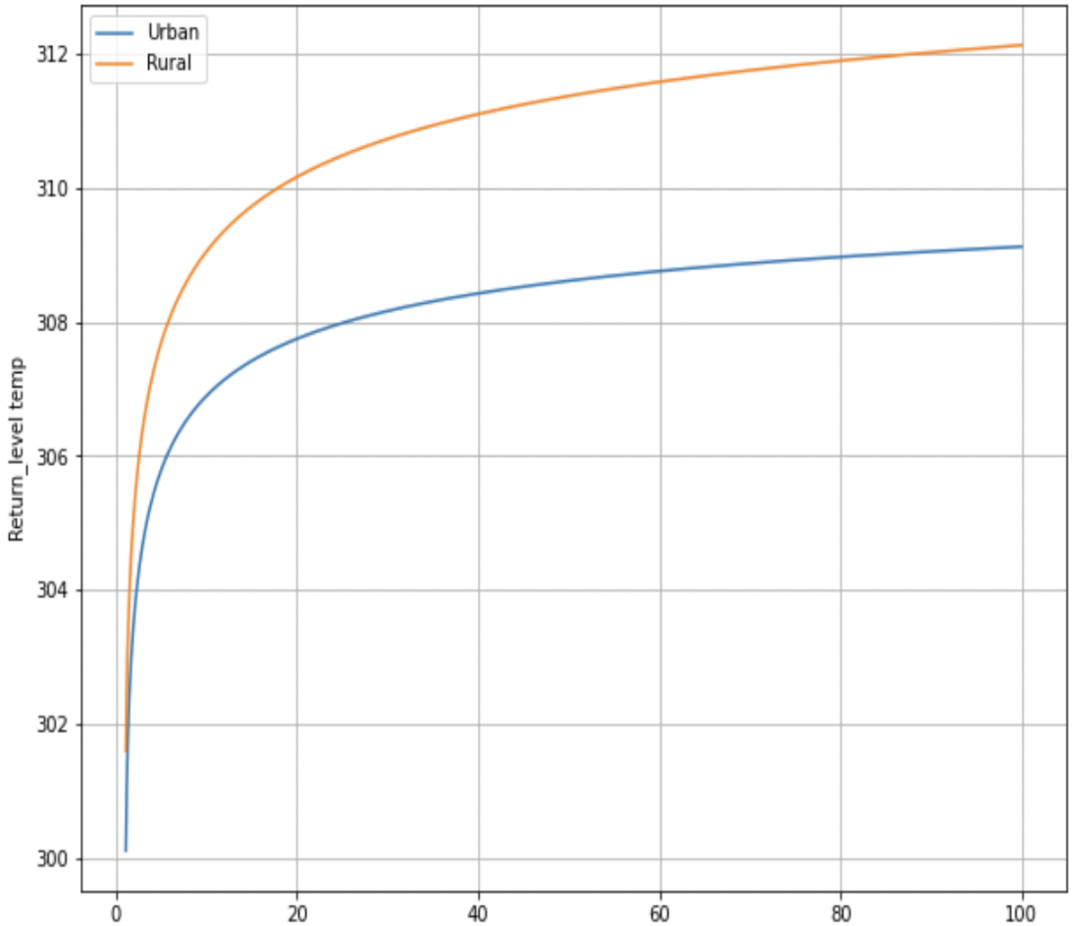
4. JJA Precipitation



Step 4:

1.Surface Air Temperature (tas)

```
30 year Return Level for Urban Area =  308.2002436471396 K
100 year Return Level  for Urban Area=  309.126110471803 K
30 year Return Level for Rural Area =  310.7837708207104 K
100 year Return Level  for Rural Area=  312.1366613291383 K
<matplotlib.legend.Legend at 0x7f384d570c90>
```

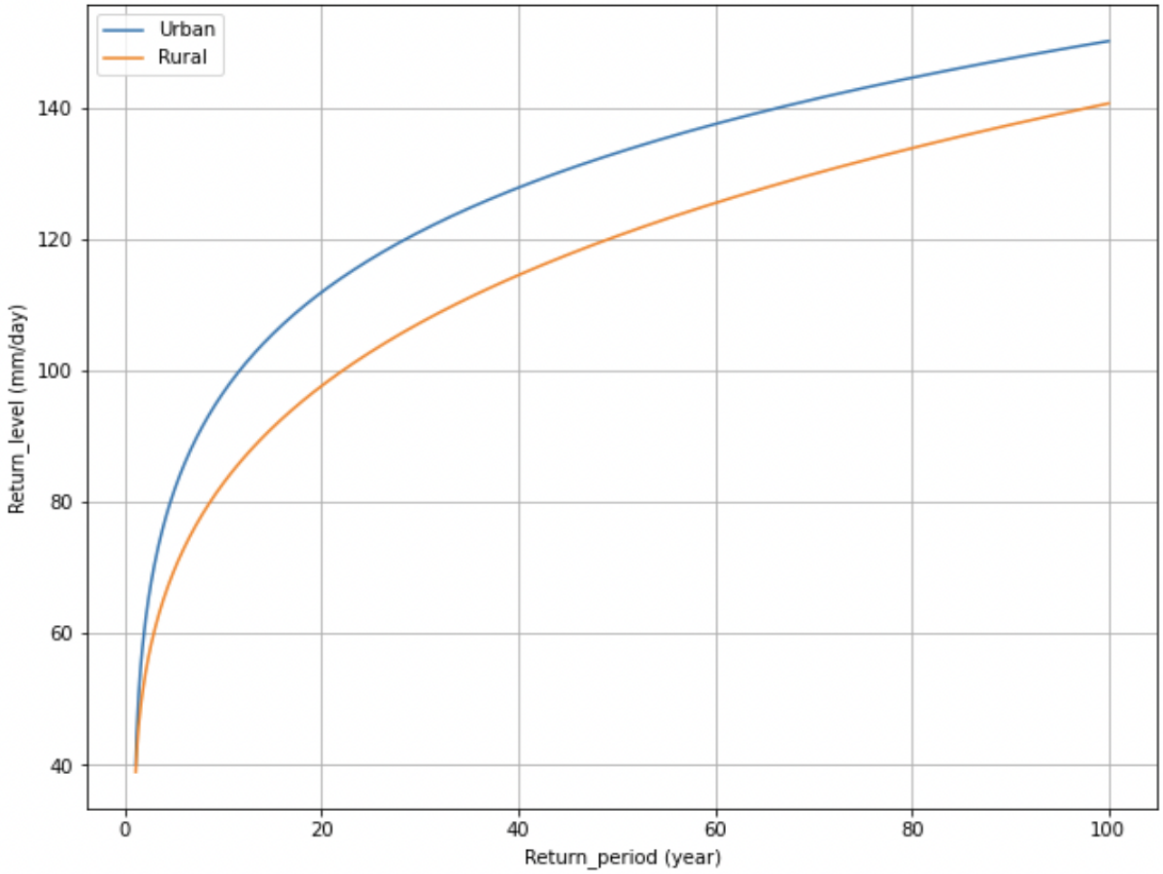


Comments:

The return levels for Amherst(rural region) are higher than that of Boston(urban region) for both 30 and 100 year return periods.

2.Precipitation

```
30 year Return Level for Urban Area =  122.03479657770986 mm/day
100 year Return Level  for Urban Area=  150.21790005522928 mm/day
30 year Return Level for Rural Area =  108.19524112228382 mm/day
100 year Return Level  for Rural Area=  140.7252819587472 mm/day
<matplotlib.legend.Legend at 0x7f225961ad10>
```



The return levels for precipitation for Amherst(rural region) are lower than that of Boston(urban region) for both 30 and 100 year return periods.

Step 5:

Urban region (Boston population): 1621281.75

Rural region (Amherst population): 25029.466796875

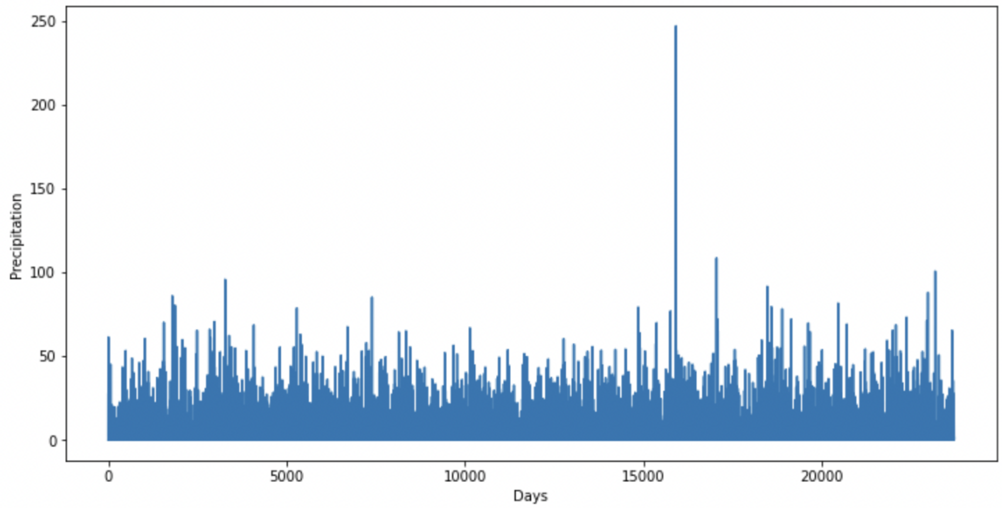
```
PEVI_Urban= 1.230943175781613
PEVI_Rural= 1.3006605512316152
```

Risks: 1. Boston(urban): 1.2079348460952744; 2. Amherst(rural):0.0130066

Based on future return levels for both climate variables, Population in Boston (urban region) is more at risk than the population in Amherst(rural region).

2. Precipitation Prediction

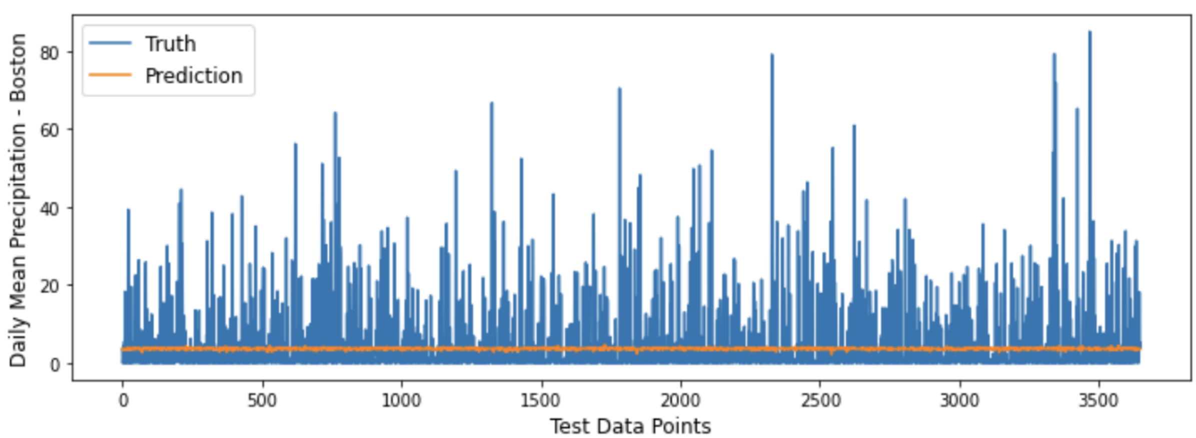
Visualization of precipitation over 65 years.



Linear model:

```
MSE:  61.253874346515964
MAE:  4.766788897551297
R2:  0.005724239536816518
```

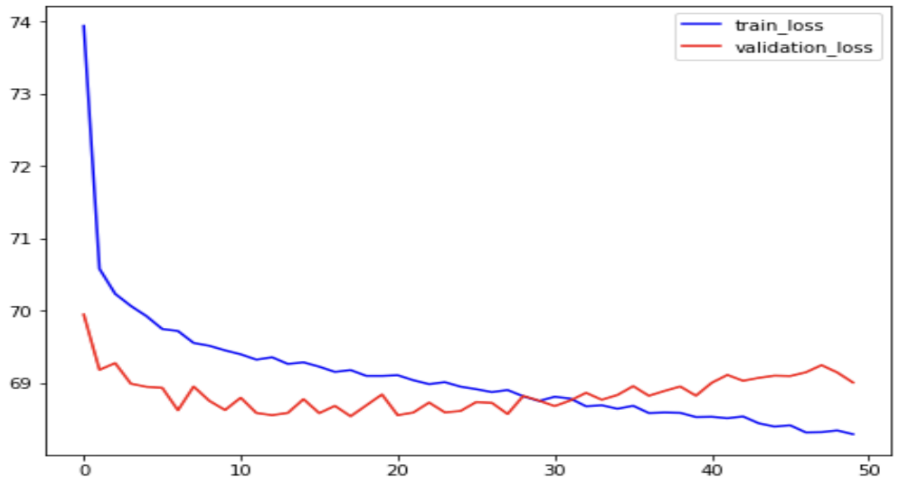
Plot



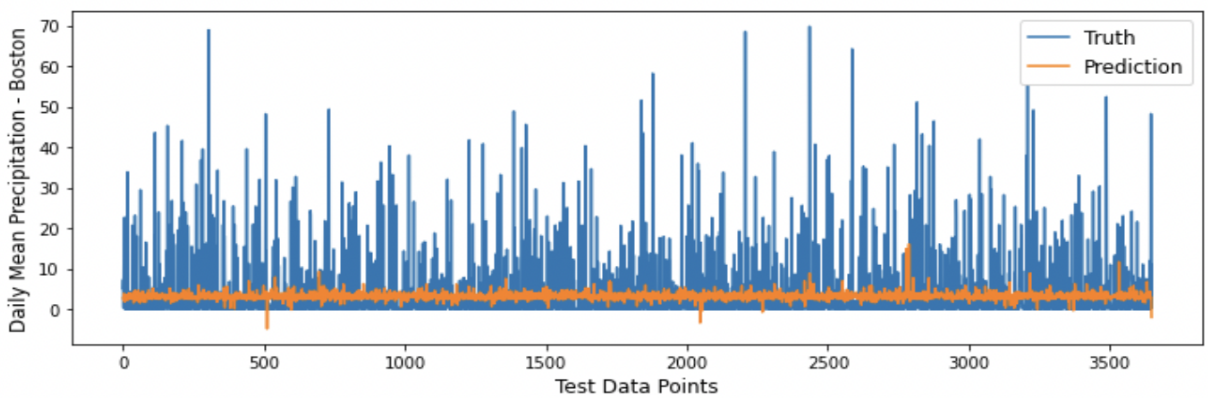
Deep learning model:

```
MSE: 61.63689422607422
MAE: 4.561830997467041
R2:  0.00026326765886997006
```

MSE Losses



Plot



Comments:

Both models - Linear and the Deep Learning model have not performed well. They both have high MSE values.

The linear model has as MSE of `61.253` and the DL model has MSE: `61.63`. This is probably because there is no observable trend in the Precipitation parameter as I observed by plotting. So despite the fact that both models do not perform well, the Linear model has an ever so slight lower MSE value than the DL Model, so it performs very slightly better than the deep learning model.

3. Literature Review

Ans 3(a) Based on the lecture by Nishant Yadav and the following paper on hyperparameter search in deep NN's write
plain language Summary of the knowledge advance and their applications
(no more than 150 words)

In later works, a few propels in neural engineering, as well as the explainability and interpretability of connectionist models, have been accomplished. The paper underlines the understanding of how to construct Bayesian Deep Learning (BDL) hyperparameters for the reason of vigorous work mapping with vulnerability evaluation, strikingly the depth, width, and gathering estimate which are still developing.The paper takes after a bottom-up approach for designing the network configuration for tests. By mapping Bayesian connectionist representations to polynomials of different orders with differing noise sorts and proportions, this research aims to make strides in readers' comprehension. The paper looks at noise-contaminated polynomials in order to discover a set of hyperparameters that can recover the fundamental polynomial signals while measuring vulnerability based on noise features. The paper points to deciding whether a satisfactory neural engineering and ensemble arrangement can be created to detect any n-th signal.

Ans 3(b)

(i)Transportation Network-of-Networks Under Compound Extremes(not more than 100words)

Network resilience is measured by the rate at which functionality is lost if a subset of nodes fails**.** Inspired by percolation theory, the GiantConnectedComponent (GCC) is treated as a proxy for the direct functionality of the network. At each discrete-time step (stopped)**,** the magnitude of GCC can be calculated from an adjacency matrix that encodes all information about the network using an appropriate algorithm, such as Kosaraju's depth-first search algorithm. The multiscale interconnected nature of MUTS, combined with the inherent unpredictability of extreme weather events, makes resilience tasks even more difficult.

(ii)Gaussian Processes for Parameter Estimation and UQ in Nonlinear Dynamical Systems (not more than 100words).

Despite their conditional Gaussianity, such systems are highly nonlinear and capable of capturing non-Gaussian characteristics in nature. The system's unique structure enables closed analytical equations to solve conditional statistics, making it computationally efficient. Data-driven physics-constrained nonlinear stochastic models, stochastically linked reaction–diffusion models in neurology and ecology, and large-scale dynamical models in turbulence, fluids, and geophysical flows are among the examples of conditional Gaussian systems shown here. It is also used to develop extremely low-cost multiscale data assimilation schemes, such as stochastic superparameterization, which uses particle filters to capture non-Gaussian statistics on the large-scale part with a small dimension.

(iii)Deep Transfer Learning for Air Quality (AQ) Estimation (Not more than 100 words)

With deep transfer learning, it is possible to estimate air quality in emerging low-resource cities with limited observations.Ex: an unsupervised transfer learning strategy (DeepAQ) for the city of Accra, Africa. The DeepAQ model follows a two-step process: First, a CNN-based model is trained to predict air quality over cities with appropriate training data at 200 m resolution (in terms of yearly average NO_2 levels). Two prospective cities have been chosen: Los Angeles and New York City. Because of the availability of labeled data and, more crucially, the large range of NO_2 levels and associated patterns, the two cities were chosen. The DeepAQ model is then transferred to Accra in an unsupervised scenario in the next step.

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
!pip install git+https://github.com/OpenHydrology/lmoments3.git
import numpy as np
import lmoments3 as lm
from lmoments3 import distr
import xarray as xr
import numpy as np
import pandas as pd
import netCDF4 as nc
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
Collecting git+https://github.com/OpenHydrology/lmoments3.git
  Cloning https://github.com/OpenHydrology/lmoments3.git to /tmp/pip-req-build-c
  Running command git clone -q https://github.com/OpenHydrology/lmoments3.git /t
```

```
fn = "/content/drive/MyDrive/Merged_1950_2019.nc"
ds =xr.open_dataset(fn)
print(ds)
```

```
<xarray.Dataset>
Dimensions:  (lat: 120, lon: 300, time: 25567)
Coordinates:
  * time     (time) datetime64[ns] 1950-01-01 1950-01-02 ... 2019-12-31
  * lat      (lat) float32 20.12 20.38 20.62 20.88 ... 49.12 49.38 49.62 49.88
  * lon      (lon) float32 230.1 230.4 230.6 230.9 ... 304.1 304.4 304.6 304.9
Data variables:
    precip   (time, lat, lon) float32 ...
```

```
ds['lon'] = ds['lon'] -360
```

```
lat=ds.lat.values
lon=ds.lon.values
```

```
precipitation = ds.sel(time = '1980-04-07')
precipitation['precip'].plot()
```

```
def gev_wrapper(data,T):
    gevfit = gev_fit(data)
    RL = return_levels(gevfit,T)

    return  RL


def gev_fit(data):

    gevfit = distr.gev.lmom_fit(data)
    return gevfit


def return_levels(gevfit,T):

    #Return Level
    RL = distr.gev.ppf(1.0-1./T, **gevfit)

    return RL



#urban area_Boston
latitude= 42.3601
longitude= -71.0589

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_urban = dsloc.groupby('time.year').max('time')


annual_max_urban
```

```
#Rural area_amherst
latitude=42.358714483009635
longitude=-72.46189157456097

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_rural = dsloc.groupby('time.year').max('time')


if not np.isnan(np.min(annual_max_urban.precip)):
    annual_max_urban=np.array(annual_max_urban.precip)
if not np.isnan(np.min(annual_max_rural.precip)):
    annual_max_rural=np.array(annual_max_rural.precip)


T_100  = np.arange(0.1, 99.1, 0.1) + 1
Urban_RL=  gev_wrapper(annual_max_urban,T_100)
Rural_RL=  gev_wrapper(annual_max_rural,T_100)



Urban_RL_30= Urban_RL[300]
Urban_RL_100= Urban_RL[-1]

Rural_RL_30= Rural_RL[300]
Rural_RL_100= Rural_RL[-1]
print(f"30 year Return Level for Urban Area =  {Urban_RL_30} mm/day")
print(f"100 year Return Level  for Urban Area=  {Urban_RL_100} mm/day")

print(f"30 year Return Level for Rural Area =  {Rural_RL_30} mm/day")
```

```
print(f"100 year Return Level  for Rural Area=  {Rural_RL_100} mm/day")

plt.figure(figsize=[10,8])
plt.plot(T_100,Urban_RL,label='Urban')
plt.plot(T_100,Rural_RL,label='Rural')
plt.grid()
plt.ylabel('Return_level (mm/day)')
plt.xlabel('Return_period (year)')
plt.legend()
```

```
PEVI_Urban= Urban_RL_100/Urban_RL_30
print(f"PEVI_Urban= {PEVI_Urban}")
PEVI_Rural= Rural_RL_100/Rural_RL_30
print(f"PEVI_Rural= {PEVI_Rural}")
```

```
PEVI_Urban= 1.230943175781613
PEVI_Rural= 1.3006605512316152
```

```
fn = "/content/drive/MyDrive/pr_day_CanESM5_ssp585_r13i1p2f1_gn_20150101-21001231.nc"
ds =xr.open_dataset(fn)
```

```
print(ds)
```

```
<xarray.Dataset>
Dimensions:    (bnds: 2, lat: 64, lon: 128, time: 31390)
Coordinates:
  * time       (time) object 2015-01-01 12:00:00 ... 2100-12-31 12:00:00
  * lat        (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon        (lon) float64 0.0 2.812 5.625 8.438 ... 348.8 351.6 354.4 357.2
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) object ...
    lat_bnds   (lat, bnds) float64 ...
    lon_bnds   (lon, bnds) float64 ...
    pr         (time, lat, lon) float32 ...
Attributes: (12/53)
    CCCma_model_hash:            f40814ae97970257f253e53802f6dcb79ec2bb26
    CCCma_parent_runid:          p2-his13
    CCCma_pycmor_hash:           26c970628162d607fffd14254956ebc6dd3b6f49
    CCCma_runid:                 p2-s8513
    Conventions:                 CF-1.7 CMIP-6.2
    YMDH_branch_time_in_child:   2015:01:01:00
    ...                          ...
    tracking_id:                 hdl:21.14100/6ebcd3d4-f81a-4f86-87b0-fa17105...
    variable_id:                 pr
    variant_label:               r13i1p2f1
    version:                     v20190429
    license:                     CMIP6 model data produced by The Government ...
    cmor_version:                3.5.0
```

```
pr = ds.sel(time = '2050-04-07')
pr['pr'].plot()
```

```
ds['lon'] = ds['lon'] -360
ds['pr'] = ds['pr'] *86400
lat=ds.lat.values
```

```
lon=ds.lon.values
#urban area_Boston
latitude= 42.3601
longitude= -71.0589

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_urban = dsloc.groupby('time.year').max('time')
#Rural area_amherst
latitude=42.358714483009635
longitude=-72.46189157456097

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_rural = dsloc.groupby('time.year').max('time')

if not np.isnan(np.min(annual_max_urban.pr)):
    annual_max_urban=np.array(annual_max_urban.pr)
if not np.isnan(np.min(annual_max_rural.pr)):
    annual_max_rural=np.array(annual_max_rural.pr)

T_100  = np.arange(0.1, 99.1, 0.1) + 1
Future_Urban_RL=  gev_wrapper(annual_max_urban,T_100)
Future_Rural_RL=  gev_wrapper(annual_max_rural,T_100)



Future_Urban_RL_30= Future_Urban_RL[300]
Future_Urban_RL_100= Future_Urban_RL[-1]

Future_Rural_RL_30= Future_Rural_RL[300]
Future_Rural_RL_100= Future_Rural_RL[-1]
print(f"30 year Return Level for Urban Area =  {Urban_RL_30} mm/hr")
print(f"100 year Return Level  for Urban Area=  {Urban_RL_100} mm/hr")
```

```
print(f"30 year Return Level for Rural Area =  {Rural_RL_30} mm/hr")
print(f"100 year Return Level  for Rural Area=  {Rural_RL_100} mm/hr")

plt.figure(figsize=[10,8])
plt.plot(T_100,Future_Urban_RL,label='Urban')
plt.plot(T_100,Future_Rural_RL,label='Rural')
plt.grid()
plt.ylabel('Return_level (mm/hr)')
plt.xlabel('Return_period (year)')
plt.legend()

PEVI_Urban= Future_Urban_RL_100/Future_Urban_RL_30
print(f"PEVI_Urban= {PEVI_Urban}")
PEVI_Rural= Future_Rural_RL_100/Future_Rural_RL_30
print(f"PEVI_Rural= {PEVI_Rural}")

PEVI_future_urban= Future_Urban_RL_100/Urban_RL_100
print(f"PEVI_future_urban= {PEVI_future_urban}")



plt.figure(figsize=[10,8])
plt.plot(T_100,Urban_RL,label='Present Urban')
plt.plot(T_100,Future_Urban_RL,label='Future Urban')
plt.grid()
plt.ylabel('Return_level (mm/hr)')
plt.xlabel('Return_period (year)')
plt.legend()
```

Population Grid

```
file_pop= "/content/drive/MyDrive/LORS-Riddhi/gpw_v4_population_count_rev11_15_min .n
ds_pop =xr.open_dataset(file_pop)
```

```
print(ds_pop)
print(ds_pop.variables)
```

```
<xarray.Dataset>
Dimensions:                                                                    (lat
Coordinates:
  * longitude                                                                  (lon
  * latitude                                                                   (lat
  * raster                                                                     (ras
Data variables:
    Population Count, v4.11 (2000, 2005, 2010, 2015, 2020): 15 arc-minutes  (ras
Attributes:
    proj4:        +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0...
    Conventions:  CF-1.4
    created_by:   R, packages ncdf4 and raster (version 2.8-4)
    date:         2018-11-16 09:57:12
Frozen({'longitude': <xarray.IndexVariable 'longitude' (longitude: 1440)>
array([-179.875, -179.625, -179.375, ...,  179.375,  179.625,  179.875])
Attributes:
    units:      degrees_east
    long_name:  longitude, 'latitude': <xarray.IndexVariable 'latitude' (latitud
array([ 89.875,  89.625,  89.375, ..., -89.375, -89.625, -89.875])
Attributes:
    units:      degrees_north
    long_name:  latitude, 'raster': <xarray.IndexVariable 'raster' (raster: 20)>
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20], dtype=int32)
Attributes:
    units:      unknown
    long_name:  raster, 'Population Count, v4.11 (2000, 2005, 2010, 2015, 2020):
[20736000 values with dtype=float32]
Attributes:
    units:      Persons
    long_name:  Population Count, v4.11 (2000, 2005, 2010, 2015, 2020): 15 ar...
    min:        [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.000e+00  0.00...
    max:        [1.11283390e+07 1.16128430e+07 1.21455970e+07 1.27324130e+07\...
```

```
#urban_area_population
lat=ds_pop.latitude.values
lon=ds_pop.longitude.values
#urban area_Boston
```

```
latitude= 42.3601
longitude= -71.0589

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds_pop.sel(latitude=latitude_grid,longitude= longitude_grid,raster=5, method='

pop_urban=dsloc['Population Count, v4.11 (2000, 2005, 2010, 2015, 2020): 15 arc-minut
print(pop_urban)
```

```
1621281.75
```

```
#Rural area_amherst
latitude=42.358714483009635
longitude=-72.46189157456097

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds_pop.sel(latitude=latitude_grid,longitude= longitude_grid,raster=5, method='

pop_rural=dsloc['Population Count, v4.11 (2000, 2005, 2010, 2015, 2020): 15 arc-minut
print(pop_rural)
```

```
25029.466796875
```

```
urban_pop_index= (1621281.75-25029.466796875)/1621281.75
urban_pop_index
```

```
0.9845619265147005
```

```
risk_in_urban_area =urban_pop_index* PEVI_Urban
risk_in_urban_area
```

```
1.2079348460952744
```

```
risk_in_rural_area = 0.01* 1.3006605512316152
```

```
risk_in_rural_area
```

```
0.013006605512316152
```

Below- Step 2,3 4-temp return levels part

```
import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

```
# Open the model dataset
# 2015 - 2054 daily surface temperature from CESM 2.0

ds1 = xr.open_dataset("/content/drive/MyDrive/LORS-Riddhi/tas_day_CanESM5_ssp585_r13i
ds1
```

```
ds
```

```
pr = ds['pr']
```

```
tas = ds1['tas']
```

```
# Plotting
temp = ds1.sel(time="2023-01-01") #select a date
temp['tas'].plot(figsize=(12,6))
plt.show()
```

```
# Finding Annual Mean

annual_mean = ds1.groupby('time.year').mean('time')
annual_mean
```

```
# Plot the change in annual mean from 2015 to 2054 for a specific place (e.g.amherst)
#amherst co ordinates
lat = 42.358714483009635
lon = 360 -72.46189157456097

boston_annual_mean = annual_mean.sel(lat=lat, lon=lon, method='nearest')
boston_annual_mean
```

```
lati = 42.3601
longi= 360 -71.0589
b_annual_mean = annual_mean.sel(lat=lati, lon=longi, method='nearest')


# Plot
boston_annual_mean['tas'].plot()
plt.show()
```

step 2 temperature part

```
# Plotting a trend line

y = boston_annual_mean['tas'].values # extract the temperature numpy array
#print(np.size(y))
x = np.arange(2014,2100) # years

coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


boston_annual_mean['tas'].plot()
plt.plot(x,trend)
plt.show()
```

```
y = b_annual_mean['tas'].values # extract the temperature numpy array
#print(np.size(y))
```

```
x = np.arange(2014,2100) # years

coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept
b_annual_mean['tas'].plot()
plt.plot(x,trend)
plt.show()
```

Step 3 DJF and JJA for temperature only

```
# Finding "annual" mean for a specific season


ds1_DJF = ds1.where(ds1['time.season'] == 'DJF') # extract data for only DJF season
ds1_JJA = ds1.where(ds1['time.season']== 'JJA')

DJF_mean = ds1_DJF.groupby('time.year').mean(dim = "time") #group by annual values an
JJA_mean = ds1_JJA.groupby('time.year').mean(dim='time')
print(DJF_mean)
print(JJA_mean)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, lat: 64, lon: 128, year: 86)
Coordinates:
  * lat       (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon       (lon) float64 0.0 2.812 5.625 8.438 ... 348.8 351.6 354.4 357.2
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
```

```
    lat_bnds  (year, lat, bnds) float64 -90.0 -86.58 -86.58 ... 86.58 86.58 90.0
    lon_bnds  (year, lon, bnds) float64 -1.406 1.406 1.406 ... 355.8 355.8 358.6
    tas       (year, lat, lon) float32 247.8 247.6 247.4 ... 275.8 275.8 275.8
<xarray.Dataset>
Dimensions:   (bnds: 2, lat: 64, lon: 128, year: 86)
Coordinates:
  * lat       (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon       (lon) float64 0.0 2.812 5.625 8.438 ... 348.8 351.6 354.4 357.2
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, lat, bnds) float64 -90.0 -86.58 -86.58 ... 86.58 86.58 90.0
    lon_bnds  (year, lon, bnds) float64 -1.406 1.406 1.406 ... 355.8 355.8 358.6
    tas       (year, lat, lon) float32 221.2 220.9 220.6 ... 279.7 279.7 279.7
```

```
# Seaonalal annual max for amherst

boston_DJF_mean = DJF_mean.sel(lat=lat, lon=lon, method='nearest')
print(boston_DJF_mean)
amherst_JJA_mean = JJA_mean.sel(lat=lat, lon=lon, method='nearest')
print(amherst_JJA_mean)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 286.9
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 285.5 288.3 285.5 288.3 ... 288.3 285.5 288.3
    tas       (year) float32 269.7 271.7 271.7 267.4 ... 278.4 276.4 278.9 278.4
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 286.9
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 285.5 288.3 285.5 288.3 ... 288.3 285.5 288.3
    tas       (year) float32 294.7 295.7 294.9 294.8 ... 301.3 302.2 302.1 301.8
```

```
lati = 42.3601
longi= 360 -71.0589
b_DJF_mean = DJF_mean.sel(lat=42.3601, lon=360 -71.0589, method='nearest')
print(b_DJF_mean)
```

```
b_JJA_mean = JJA_mean.sel(lat=42.3601, lon=360 -71.0589, method='nearest')
print(b_JJA_mean)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 289.7
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 288.3 291.1 288.3 291.1 ... 291.1 288.3 291.1
    tas       (year) float32 275.3 276.6 277.1 274.3 ... 283.4 282.5 283.8 283.5
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 289.7
    height    float64 2.0
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 288.3 291.1 288.3 291.1 ... 291.1 288.3 291.1
    tas       (year) float32 294.5 295.3 294.8 295.2 ... 301.4 302.8 301.9 302.0
```

- **DJF temp *plot**

```
boston_DJF_mean['tas'].plot()
plt.show()
```

```
b_DJF_mean['tas'].plot()
plt.show()
```

JJA temp plot

```
amherst_JJA_mean['tas'].plot()
plt.show()
```

```
b_JJA_mean['tas'].plot()
plt.show()
```

```
#mean temperature
y = boston_DJF_mean['tas'].values # extract the temperature numpy array

x = np.arange(2014,2100) # years

coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


#mean temperature
y = b_DJF_mean['tas'].values # extract the temperature numpy array

x = np.arange(2014,2100) # years

coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


#mean temperature
y = b_JJA_mean['tas'].values # extract the temperature numpy array

x = np.arange(2014,2100) # years

coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


#mean temperature
y = amherst_JJA_mean['tas'].values # extract the temperature numpy array

x = np.arange(2014,2100) # years
```

```
coefficients = np.polyfit(x,y,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


boston_DJF_mean['tas'].plot()
plt.plot(x, trend)
plt.show()
```

```
amherst_JJA_mean['tas'].plot()
plt.plot(x, trend)
plt.show()
```

```
b_DJF_mean['tas'].plot()
```

```
plt.plot(x, trend)
plt.show()
```

```
b_JJA_mean['tas'].plot()
plt.plot(x, trend)
plt.show()
```

*Step 2- for precipitation variable *

```
# Plotting
prec = ds.sel(time="2023-01-01") #select a date
prec['pr'].plot(figsize=(12,6))
plt.show()
```

```
# Finding Annual Mean

annual_mean_prec = ds.groupby('time.year').mean('time')
annual_mean_prec
```

```
# Plot the change in annual mean from 2015 to 2054 for a specific place (e.g. amherst

lat = 42.358714483009635
lon = 360 -72.46189157456097

amherst_annual_mean_prec = annual_mean_prec.sel(lat=lat, lon=lon, method='nearest')
amherst_annual_mean_prec
```

```
#boston
latii= 42.3601
longii= 360 -71.0589

kore_annual_mean_prec = annual_mean_prec.sel(lat=latii, lon=longii, method='nearest')
kore_annual_mean_prec
```

```
# Plotting a trend line

b = amherst_annual_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years

coefficients = np.polyfit(a,b,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept
```

```
# Plotting a trend line

b = kore_annual_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years
```

```
coefficients = np.polyfit(a,b,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


kore_annual_mean_prec['pr'].plot()
plt.plot(a,trend)
plt.show()
```

AMherst **precipitation** projection plot

```
amherst_annual_mean_prec['pr'].plot()
plt.plot(a,trend)
plt.show()
```

Step 3

DDJ and JJA for precipitation

```
# Finding "annual" mean for a specific season


ds_DJF = ds.where(ds['time.season'] == 'DJF') # extract data for only DJF season
ds_JJA = ds.where(ds['time.season'] == 'JJA')

DJF_mean_prec = ds_DJF.groupby('time.year').mean(dim = "time") #group by annual value
JJA_mean_prec = ds_JJA.groupby('time.year').mean(dim='time')
print(DJF_mean_prec)
print(JJA_mean_prec)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, lat: 64, lon: 128, year: 86)
Coordinates:
  * lat       (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon       (lon) float64 -360.0 -357.2 -354.4 -351.6 ... -8.438 -5.625 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, lat, bnds) float64 -90.0 -86.58 -86.58 ... 86.58 86.58 90.0
    lon_bnds  (year, lon, bnds) float64 -1.406 1.406 1.406 ... 355.8 355.8 358.6
    pr        (year, lat, lon) float64 0.3637 0.3587 0.3728 ... 1.361 1.421
<xarray.Dataset>
Dimensions:   (bnds: 2, lat: 64, lon: 128, year: 86)
Coordinates:
  * lat       (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon       (lon) float64 -360.0 -357.2 -354.4 -351.6 ... -8.438 -5.625 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, lat, bnds) float64 -90.0 -86.58 -86.58 ... 86.58 86.58 90.0
    lon_bnds  (year, lon, bnds) float64 -1.406 1.406 1.406 ... 355.8 355.8 358.6
    pr        (year, lat, lon) float64 0.1256 0.1205 0.1165 ... 1.883 1.923
```

```
mybos_DJF_mean_prec = DJF_mean_prec.sel(lat=42.3601, lon=360 -71.0589, method='neares
print(mybos_DJF_mean_prec)
mybos_JJA_mean_prec = JJA_mean_prec.sel(lat=42.3601, lon=360 -71.0589, method='neares
print(mybos_JJA_mean_prec)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
```

```
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 2.83 4.221 1.829 2.418 ... 2.267 2.373 2.866 2.815
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 1.082 1.506 0.6261 2.76 ... 0.3217 0.2844 0.1998
```

```
# Seaonalal annual max for amherst

amherst_DJF_mean_prec = DJF_mean_prec.sel(lat=lat, lon=lon, method='nearest')
print(amherst_DJF_mean_prec)
amherst_JJA_mean_prec = JJA_mean_prec.sel(lat=lat, lon=lon, method='nearest')
print(amherst_JJA_mean_prec)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 2.83 4.221 1.829 2.418 ... 2.267 2.373 2.866 2.815
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 1.082 1.506 0.6261 2.76 ... 0.3217 0.2844 0.1998
```

```
# Seaonalal annual max for amherst

kore_DJF_mean_prec = DJF_mean_prec.sel(lat=latii, lon=longii, method='nearest')
print(kore_DJF_mean_prec)
```

```
kore_JJA_mean_prec = JJA_mean_prec.sel(lat=latii, lon=longii, method='nearest')
print(kore_JJA_mean_prec)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 2.83 4.221 1.829 2.418 ... 2.267 2.373 2.866 2.815
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 1.082 1.506 0.6261 2.76 ... 0.3217 0.2844 0.1998
```

```
b = amherst_DJF_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years

coefficients = np.polyfit(a,b,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


bosprec = kore_DJF_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years

coefficients = np.polyfit(a,bosprec,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


bosprec1 = kore_JJA_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years
```

```
coefficients = np.polyfit(a,bosprec1,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept
```

```
kore_DJF_mean_prec['pr'].plot()
plt.plot(a, trend)
plt.show()
```

```
kore_JJA_mean_prec['pr'].plot()
plt.plot(a, trend)
plt.show()
```

```
b = amherst_JJA_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years
```

```
coefficients = np.polyfit(a,b,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


l = 42.3601
lo = 360 -71.0589

#boston
latitude= 42.3601
longitude= -71.0589

mybos1_DJF_mean_prec = DJF_mean_prec.sel(lat=l, lon=lo, method='nearest')
print(mybos1_DJF_mean_prec)
mybos1_JJA_mean_prec = JJA_mean_prec.sel(lat=l, lon=lo, method='nearest')
print(mybos1_JJA_mean_prec)
```

```
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 2.83 4.221 1.829 2.418 ... 2.267 2.373 2.866 2.815
<xarray.Dataset>
Dimensions:   (bnds: 2, year: 86)
Coordinates:
    lat       float64 43.25
    lon       float64 -2.812
  * year      (year) int64 2015 2016 2017 2018 2019 ... 2096 2097 2098 2099 2100
Dimensions without coordinates: bnds
Data variables:
    lat_bnds  (year, bnds) float64 41.86 44.66 41.86 44.66 ... 44.66 41.86 44.66
    lon_bnds  (year, bnds) float64 355.8 358.6 355.8 358.6 ... 358.6 355.8 358.6
    pr        (year) float64 1.082 1.506 0.6261 2.76 ... 0.3217 0.2844 0.1998
```

```
p = mybos1_DJF_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years

coefficients2 = np.polyfit(a,p,1)

coefficients2
slope1 = coefficients2[0]
```

```
intercept1 = coefficients2[1]

trend2 = slope1*a + intercept1

mybos1_DJF_mean_prec['pr'].plot()
plt.plot(a, trend2)
plt.show()
```

```
b = amherst_JJA_mean_prec['pr'].values # extract the temperature numpy array
a = np.arange(2014,2100) # years

coefficients = np.polyfit(a,b,1)

coefficients
slope = coefficients[0]
intercept = coefficients[1]

trend = slope*x + intercept


amherst_JJA_mean_prec['pr'].plot()
plt.plot(x, trend)
plt.show()
```

```
amherst_DJF_mean_prec['pr'].plot()
plt.plot(x, trend)
plt.show()
```

DJF seasonal precipitation plot

```
amherst_DJF_mean_prec['pr'].plot()
plt.show()
```

JJA seasonal precipitation plot

```
amherst_JJA_mean_prec['pr'].plot()
```

```
plt.show()
```

Step 4 for temperature part- return levels (for amherst) Note- make sure to change the lat, long for precipitatin as well

```
ds1['lon'] = ds1['lon'] -360
ds1['tas'] = ds1['tas'] *86400
lat=ds1.lat.values
lon=ds1.lon.values
#boston
latitude= 42.3601
longitude= -71.0589

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_urban = dsloc.groupby('time.year').max('time')
#Rural area_amherst
latitude=42.358714483009635
longitude=-72.46189157456097

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
```

```
    latitude_grid= lat.__getitem__(min_index_lat)
    longitude_grid=lon.__getitem__(min_index_lon)

    dsloc= ds.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
    annual_max_rural = dsloc.groupby('time.year').max('time')

    if not np.isnan(np.min(annual_max_urban.pr)):
        annual_max_urban=np.array(annual_max_urban.pr)
    if not np.isnan(np.min(annual_max_rural.pr)):
        annual_max_rural=np.array(annual_max_rural.pr)

    T_100  = np.arange(0.1, 99.1, 0.1) + 1
    Future_Urban_RL=  gev_wrapper(annual_max_urban,T_100)
    Future_Rural_RL=  gev_wrapper(annual_max_rural,T_100)



    Future_Urban_RL_30= Future_Urban_RL[300]
    Future_Urban_RL_100= Future_Urban_RL[-1]

    Future_Rural_RL_30= Future_Rural_RL[300]
    Future_Rural_RL_100= Future_Rural_RL[-1]
    print(f"30 year Return Level for Urban Area =  {Urban_RL_30} mm/hr")
    print(f"100 year Return Level  for Urban Area=  {Urban_RL_100} mm/hr")

    print(f"30 year Return Level for Rural Area =  {Rural_RL_30} mm/hr")
    print(f"100 year Return Level  for Rural Area=  {Rural_RL_100} mm/hr")

    plt.figure(figsize=[10,8])
    plt.plot(T_100,Future_Urban_RL,label='Urban')
    plt.plot(T_100,Future_Rural_RL,label='Rural')
    plt.grid()
    plt.ylabel('Return_level (mm/hr)')
    plt.xlabel('Return_period (year)')
    plt.legend()

    PEVI_Urban= Future_Urban_RL_100/Future_Urban_RL_30
    print(f"PEVI_Urban= {PEVI_Urban}")
    PEVI_Rural= Future_Rural_RL_100/Future_Rural_RL_30
    print(f"PEVI_Rural= {PEVI_Rural}")

    PEVI_future_urban= Future_Urban_RL_100/Urban_RL_100
    print(f"PEVI_future_urban= {PEVI_future_urban}")
```

✓ 4s completed at 7:29 PM

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
%cd /content/drive/My Drive/Colab Notebooks/
```

```
/content/drive/My Drive/Colab Notebooks
```

```
!pip install git+https://github.com/OpenHydrology/lmoments3.git
import numpy as np
import lmoments3 as lm
from lmoments3 import distr
import xarray as xr
import numpy as np
import pandas as pd
import netCDF4 as nc
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
Collecting git+https://github.com/OpenHydrology/lmoments3.git
  Cloning https://github.com/OpenHydrology/lmoments3.git to /tmp/pip-req-build-g
  Running command git clone -q https://github.com/OpenHydrology/lmoments3.git /t
```

```
fn = "/content/drive/MyDrive/LORS-Riddhi/tas_day_CanESM5_ssp585_r13i1p2f1_gn_20150101
ds1 =xr.open_dataset(fn)
print(ds1)
```

```
<xarray.Dataset>
Dimensions:    (bnds: 2, lat: 64, lon: 128, time: 31390)
Coordinates:
  * time       (time) object 2015-01-01 12:00:00 ... 2100-12-31 12:00:00
  * lat        (lat) float64 -87.86 -85.1 -82.31 -79.53 ... 82.31 85.1 87.86
  * lon        (lon) float64 0.0 2.812 5.625 8.438 ... 348.8 351.6 354.4 357.2
    height     float64 ...
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) object ...
    lat_bnds   (lat, bnds) float64 ...
    lon_bnds   (lon, bnds) float64 ...
    tas        (time, lat, lon) float32 ...
Attributes: (12/53)
    CCCma_model_hash:            f40814ae97970257f253e53802f6dcb79ec2bb26
```

```
        CCCma_parent_runid:          p2-his13
        CCCma_pycmor_hash:           26c970628162d607fffd14254956ebc6dd3b6f49
        CCCma_runid:                 p2-s8513
        Conventions:                 CF-1.7 CMIP-6.2
        YMDH_branch_time_in_child:   2015:01:01:00
        ...                          ...
        tracking_id:                 hdl:21.14100/6940c3dc-f2e8-4734-8be9-b90bfaf...
        variable_id:                 tas
        variant_label:               r13i1p2f1
        version:                     v20190429
        license:                     CMIP6 model data produced by The Government ...
        cmor_version:                3.5.0
```

```
ds1['lon'] = ds1['lon'] -360

lat=ds1.lat.values
lon=ds1.lon.values
```

```
temp = ds1.sel(time = '2002-04-07')
temp['tas'].plot()
```

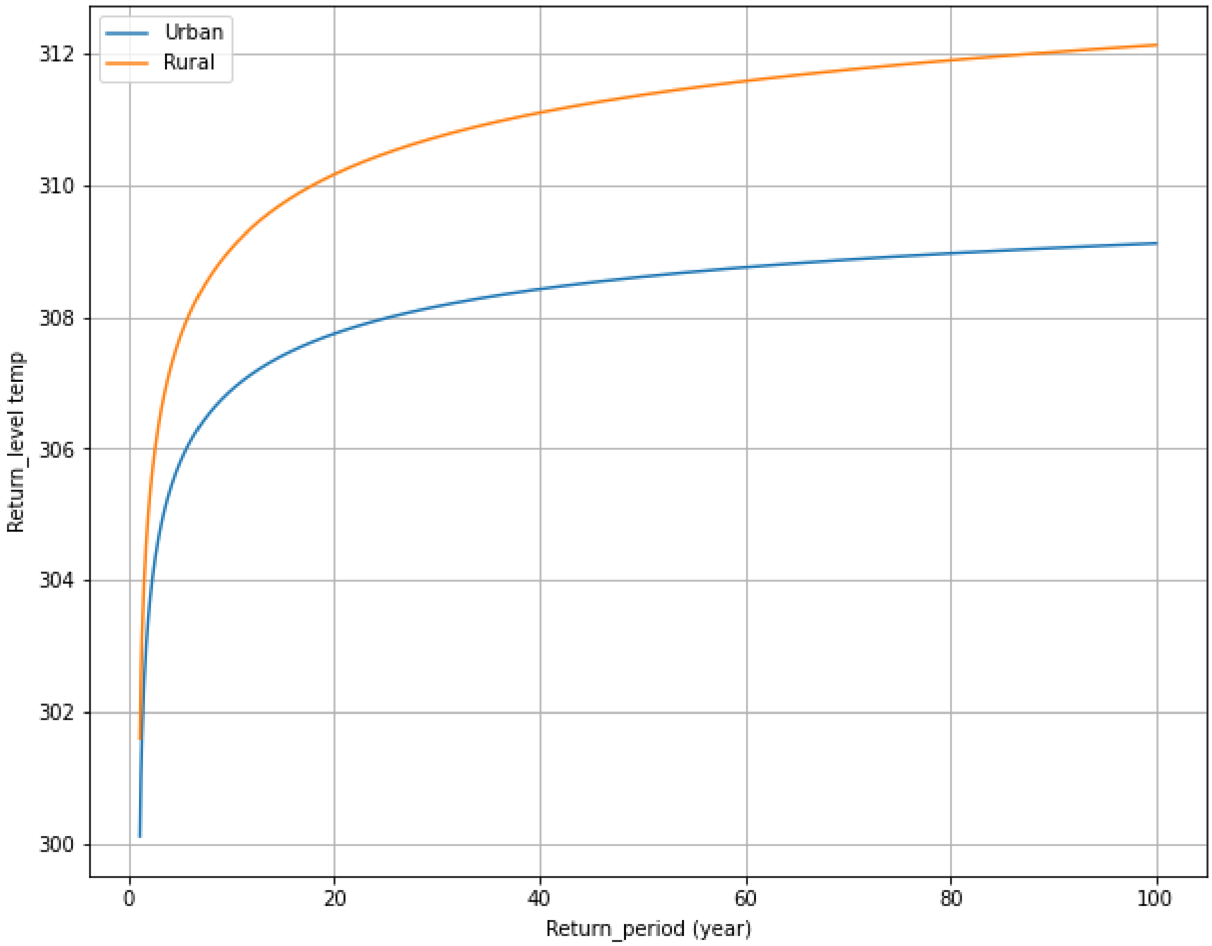
```
def gev_wrapper(data,T):
    gevfit = gev_fit(data)
    RL = return_levels(gevfit,T)

    return  RL
```

```
def gev_fit(data):

    gevfit = distr.gev.lmom_fit(data)
    return gevfit


def return_levels(gevfit,T):

    #Return Level
    RL = distr.gev.ppf(1.0-1./T, **gevfit)

    return RL


#urban area Los Angeles
#latitude=34.091840166129465
#longitude= -118.22775788492291

#urban area_Boston
latitude= 42.3601
longitude= -71.0589

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds1.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_urban = dsloc.groupby('time.year').max('time')


annual_max_urban
```

```
#amherst
latitude=42.358714483009635
longitude=-72.46189157456097

#Claremont
#latitude=34.12063925677254
#longitude= -117.67478972902865

sq_diff_lat = (lat - latitude)**2
sq_diff_lon = (lon - longitude)**2

#Identify the index of the min value for lat and lon
min_index_lat = sq_diff_lat.argmin()
min_index_lon = sq_diff_lon.argmin()
latitude_grid= lat.__getitem__(min_index_lat)
longitude_grid=lon.__getitem__(min_index_lon)

dsloc= ds1.sel(lat=latitude_grid,lon= longitude_grid, method='nearest')
annual_max_rural = dsloc.groupby('time.year').max('time')


#non considering null values
if not np.isnan(np.min(annual_max_urban.tas)):
    annual_max_urban=np.array(annual_max_urban.tas)
if not np.isnan(np.min(annual_max_rural.tas)):
    annual_max_rural=np.array(annual_max_rural.tas)


T_100  = np.arange(0.1, 99.1, 0.1) + 1
Urban_RL=  gev_wrapper(annual_max_urban,T_100)
Rural_RL=  gev_wrapper(annual_max_rural,T_100)



Urban_RL_30= Urban_RL[300]
Urban_RL_100= Urban_RL[-1]

Rural_RL_30= Rural_RL[300]
Rural_RL_100= Rural_RL[-1]
print(f"30 year Return Level for Urban Area =  {Urban_RL_30} K")
print(f"100 year Return Level  for Urban Area=  {Urban_RL_100} K")

print(f"30 year Return Level for Rural Area =  {Rural_RL_30} K")
print(f"100 year Return Level  for Rural Area=  {Rural_RL_100} K")

plt.figure(figsize=[10,8])
plt.plot(T_100,Urban_RL,label='Urban')
```

```
plt.plot(T_100,Rural_RL,label='Rural')
plt.grid()
plt.ylabel('Return_level temp')
plt.xlabel('Return_period (year)')
plt.legend()
```

```
30 year Return Level for Urban Area =  308.2002436471396 K
100 year Return Level  for Urban Area=  309.126110471803 K
30 year Return Level for Rural Area =  310.7837708207104 K
100 year Return Level  for Rural Area=  312.1366613291383 K
<matplotlib.legend.Legend at 0x7f384d570c90>
```



✓ 0s completed at 7:21 PM