import warnings

import itertools

import numpy as np

import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

plt.style.use('fivethirtyeight')

import pandas as pd

import statsmodels.api as sm

import matplotlib

import seaborn

matplotlib.rcParams['axes.labelsize'] = 14

matplotlib.rcParams['xtick.labelsize'] = 12

matplotlib.rcParams['ytick.labelsize'] = 12

matplotlib.rcParams['text.color'] = 'k'

*//Reading Data for Bihar for MAY*

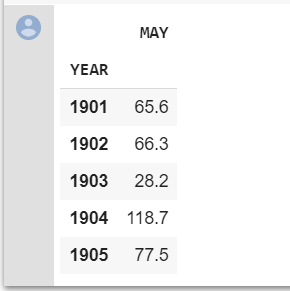
*df = pd.read\_csv("/content/drive/My Drive/Colab Notebooks/test\_data.csv")*

*avg\_rain= df.loc[df['SUBDIVISION']=='Bihar']*

*avg\_rain= avg\_rain[['YEAR','MAY']]*

*avg\_rain= avg\_rain.set\_index(['YEAR'])*

*avg\_rain.head(5)*

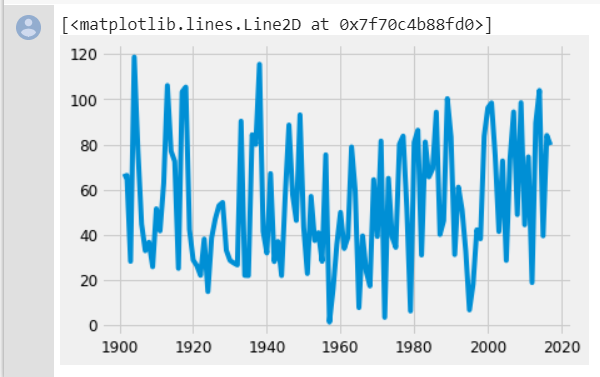
**

**Code and subsequent output**

*#Plotting the average rainfall*

*plt.plot(avg\_rain)*

**Evaluation and Results**

**

**Code and subsequent output**

#calculating rolling average with a window of 12 months

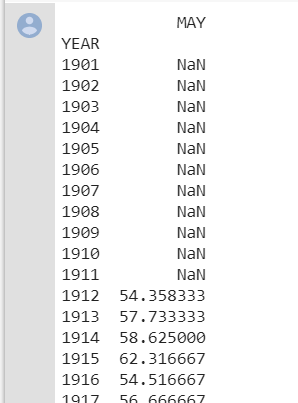
*#rolling stats for stationarity*

*rol1= avg\_rain.rolling(window=12).mean()*

*rol2= avg\_rain.rolling(window=12).std()*

*print (rol1,rol2)*

**Evaluation and Results**

*[117 rows x 1 columns]*

**Code and subsequent output**

*#plotting mean and std deviation*

#*For time series, we need a stationary mean, without which prediction won’t work. Hence, we check if mean is stationary or not. So, we calculate mean and see if it’s stationary*

*orig= plt.plot(avg\_rain,color="red",label="original")*

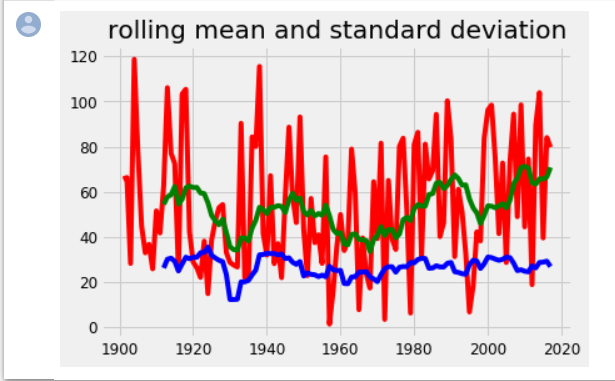
*avg= plt.plot(rol1,color="green",label="mean")*

*std= plt.plot(rol2,color="blue",label="std")*

*plt.title("rolling mean and standard deviation")*

*plt.show(block=False)*

**Evaluation and Results**

**

**Code and subsequent output**

*#Dickie Fuller Test is used to check stationarity of a particular dataset*

*from statsmodels.tsa.stattools import adfuller*

*print("DF test")*

*dftest= adfuller(avg\_rain['MAY'],autolag="AIC")*

*dfop= pd.Series(dftest[0:4],index=["test stats","p-value","lags used","no of observations used"])*

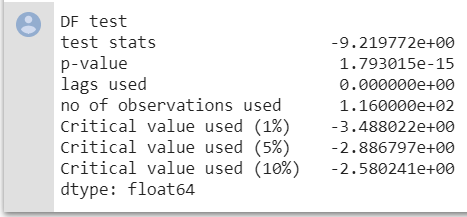
*for key,value in dftest[4].items():*

*dfop["Critical value used (%s)"%key]= value*

*print (dfop)*

*#proves that data is non stationary*

**Evaluation and Results**

**

*#we try to reduce the critical value using log or normalized version of the same graph*

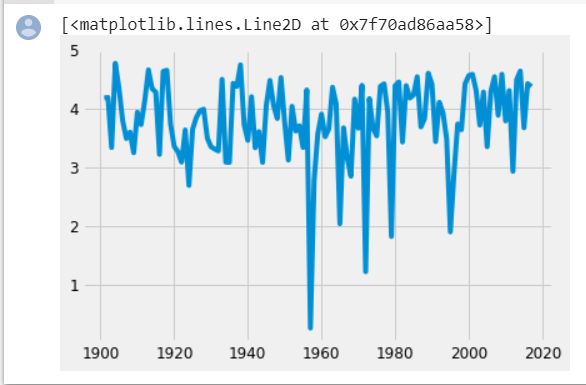
**Code and subsequent output**

#trying with the log graph

*avg\_rainlog= np.log(avg\_rain)*

*plt.plot(avg\_rainlog)*

**Evaluation and Results**

**

**Code and subsequent output**

*#same steps as for the initial graph*

*mrol1= avg\_rainlog.rolling(window=12).mean()*

*mrol2= avg\_rainlog.rolling(window=12).std()*

*#plotting mean and std deviation for moving average*

*orig= plt.plot(avg\_rainlog,color="red",label="original")*

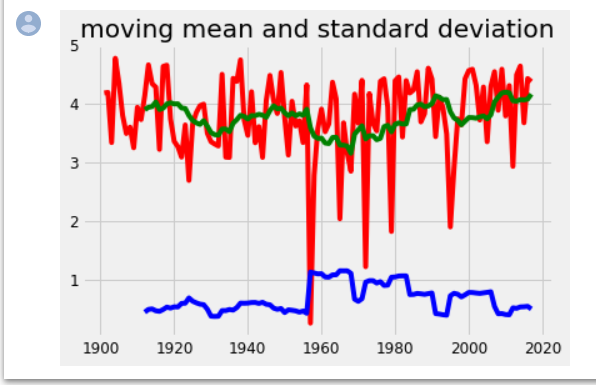
*avg= plt.plot(mrol1,color="green",label="mean")*

*std= plt.plot(mrol2,color="blue",label="std")*

*plt.title("moving mean and standard deviation")*

*plt.show(block=False)*

**Evaluation and Results**

**

**Code and subsequent output**

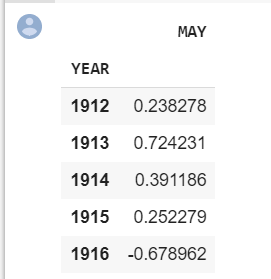
*#trying for a normalized version*

*normalized\_avg\_rain= avg\_rainlog - mrol1*

*normalized\_avg\_rain.dropna(inplace=True)*

*normalized\_avg\_rain.head()*

**Evaluation and Results**

**

**Code and subsequent output**

*#function for calculating mean and sd and then applying the DF test for any graph*

*from statsmodels.tsa.stattools import adfuller*

*def calculate(test):*

*mrol1= test.rolling(window=12).mean()*

*mrol2= test.rolling(window=12).std()*

*#plotting mean and std deviation for moving average*

*orig= plt.plot(test,color="red",label="original")*

*avg= plt.plot(mrol1,color="green",label="mean")*

*std= plt.plot(mrol2,color="blue",label="std")*

*plt.title("moving mean and standard deviation")*

*plt.show(block=False)*

*print("DF test")*

*dftest= adfuller(test['MAY'],autolag="AIC")*

*dfop= pd.Series(dftest[0:4],index=["test stats","p-value","lags used","no of observations used"])*

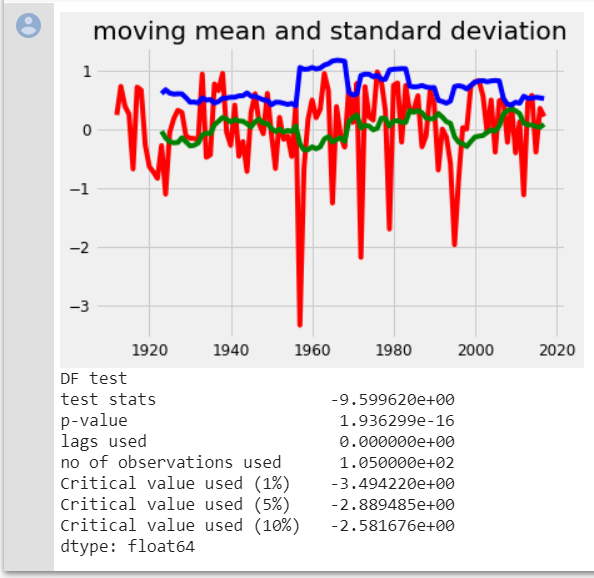
*for key,value in dftest[4].items():*

*dfop["Critical value used (%s)"%key]= value*

*print (dfop)*

*calculate(normalized\_avg\_rain)*

**Evaluation and Results**

**

**Code and subsequent output**

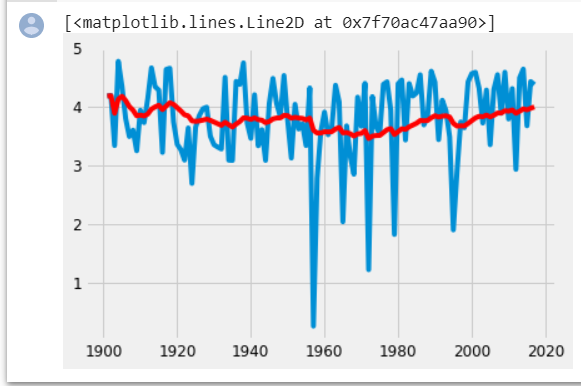
*#trying for log graph by removing decay*

*edecay= avg\_rainlog.ewm(halflife=12,min\_periods=0,adjust=True).mean()*

*plt.plot(avg\_rainlog)*

*plt.plot(edecay,color="red")*

**Evaluation and Results**

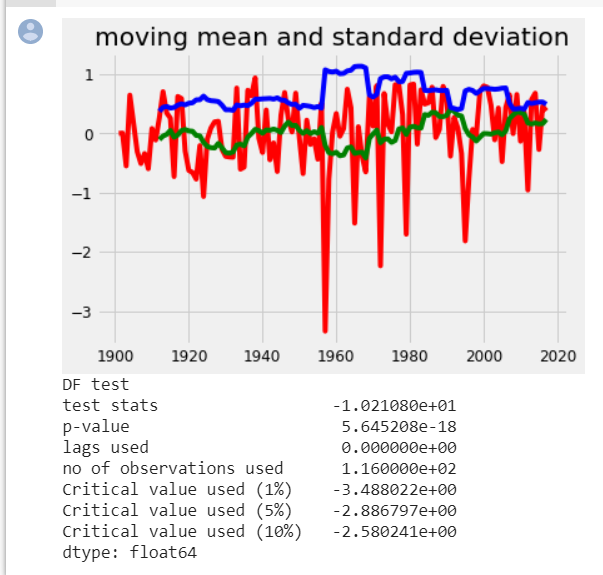
**

**Code and subsequent output**

*avg\_rain\_decay= avg\_rainlog - edecay*

*calculate (avg\_rain\_decay)*

**Evaluation and Results**

**

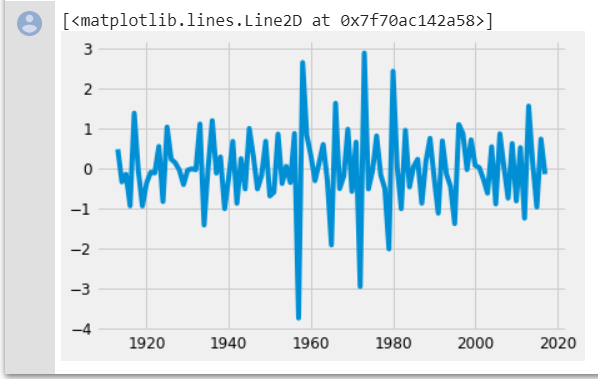
**Code and subsequent output**

*#diffrentiating the normalized graph once*

*ds\_logdiff\_shift= normalized\_avg\_rain - normalized\_avg\_rain.shift()*

*plt.plot(ds\_logdiff\_shift)*

**Evaluation and Results**

**

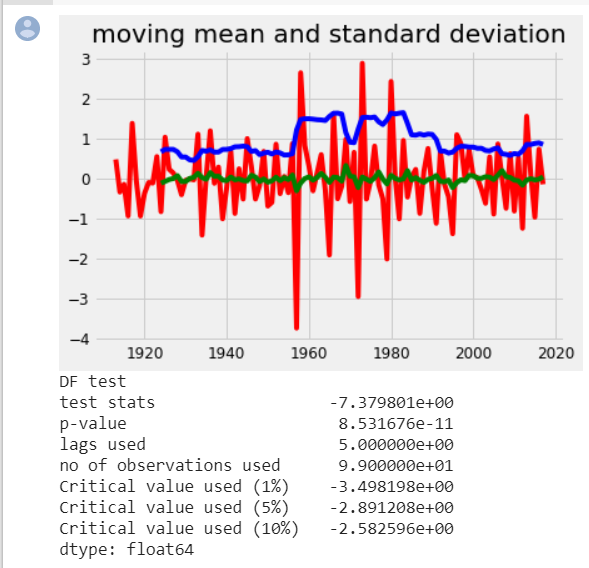
*ds\_logdiff\_shift.dropna(inplace=True)*

**Code and subsequent output**

*#DF test for the diffrentiated normalized graph*

*calculate(ds\_logdiff\_shift)*

**Evaluation and Results**

**

**Code and subsequent output**

*#we choose the optimum graph and then calculate the auto correlation and partial correlation function for the ARIMA model*

*from statsmodels.tsa.stattools import acf , pacf*

*lag\_acf= acf(ds\_logdiff\_shift,nlags=20)*

*lag\_pacf= pacf(ds\_logdiff\_shift,nlags=20,method="ols")*

*#ACF plot*

*plt.subplot(211)*

*plt.plot(lag\_acf)*

*plt.axhline (y=0,linestyle="--",color="gray")*

*plt.axhline (y=-1.96/np.sqrt(len(ds\_logdiff\_shift)),linestyle="--",color="gray")*

*plt.axhline (y=1.96/np.sqrt(len(ds\_logdiff\_shift)),linestyle="--",color="gray")*

*plt.title("ACF")*

*#PACF plot*

*plt.subplot(212)*

*plt.plot(lag\_pacf)*

*plt.axhline (y=0,linestyle="--",color="gray")*

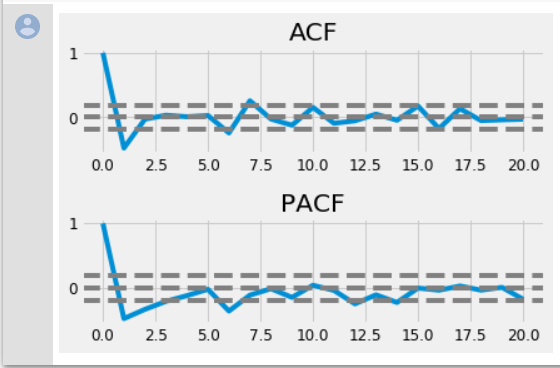
*plt.axhline (y=-1.96/np.sqrt(len(ds\_logdiff\_shift)),linestyle="--",color="gray")*

*plt.axhline (y=1.96/np.sqrt(len(ds\_logdiff\_shift)),linestyle="--",color="gray")*

*plt.title("PACF")*

*plt.tight\_layout()*

**Evaluation and Results**

**

**Code and subsequent output**

*#We input the obtained d,a,and p values into the arima model for prediction*

*Where d is the no of times diffrentiation is done, a is acf value and p is pacf value*

*from statsmodels.tsa.arima\_model import ARIMA*

*model= ARIMA(normalized\_avg\_rain, order=(1,1,0))*

*results\_AR= model.fit(disp=-1)*

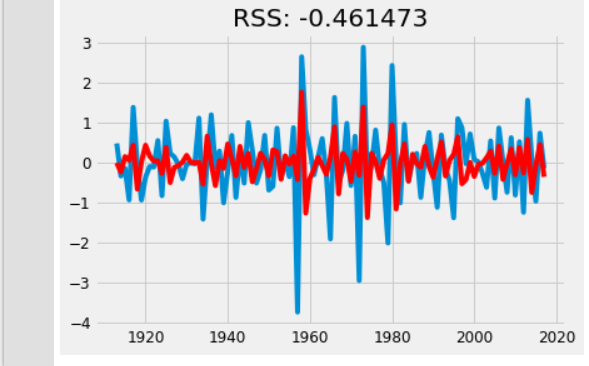
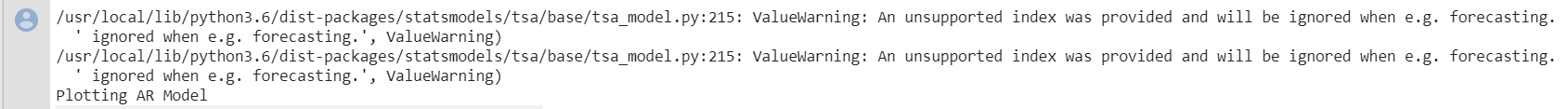
*plt.plot(ds\_logdiff\_shift)*

*plt.plot(results\_AR.fittedvalues,color="red")*

*plt.title('RSS: %4f'%sum((results\_AR.fittedvalues-ds\_logdiff\_shift['MAY'])\*2))*

*print("Plotting AR Model")*

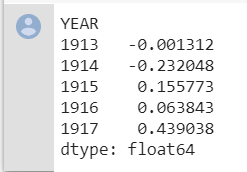
**Evaluation and Results**

**

*#The RSS value obtained is satisfactory and hence we can proceed with the prediction and we can see that the predicted model is similar*

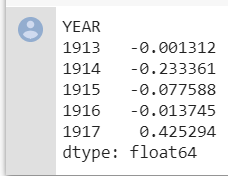
*predictions\_diff= pd.Series(results\_AR.fittedvalues,copy=True)*

*print(predictions\_diff.head())*

**

*predictions\_diff\_cum= predictions\_diff.cumsum()*

*print(predictions\_diff\_cum.head())*

**

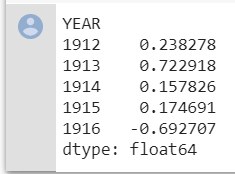
**Code and subsequent output**

**#We can see the original and the predicted model superimposed together**

*prediction\_ARIMA\_Log= pd.Series(normalized\_avg\_rain['MAY'],index= normalized\_avg\_rain.index)*

*prediction\_ARIMA\_Log= prediction\_ARIMA\_Log.add(predictions\_diff\_cum,fill\_value=0)*

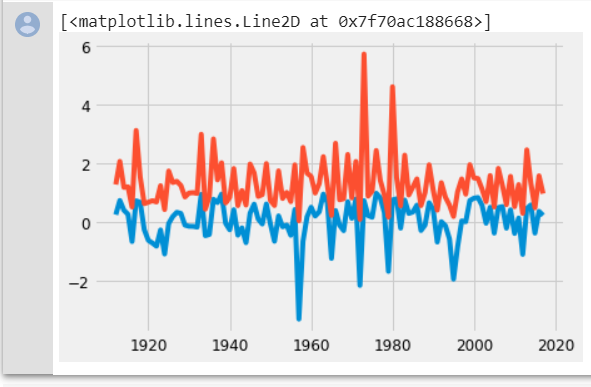
*print(prediction\_ARIMA\_Log.head())*

**

*predict= np.exp(prediction\_ARIMA\_Log)*

*plt.plot(normalized\_avg\_rain)*

*plt.plot(predict)*

**

*normalized\_avg\_rain*

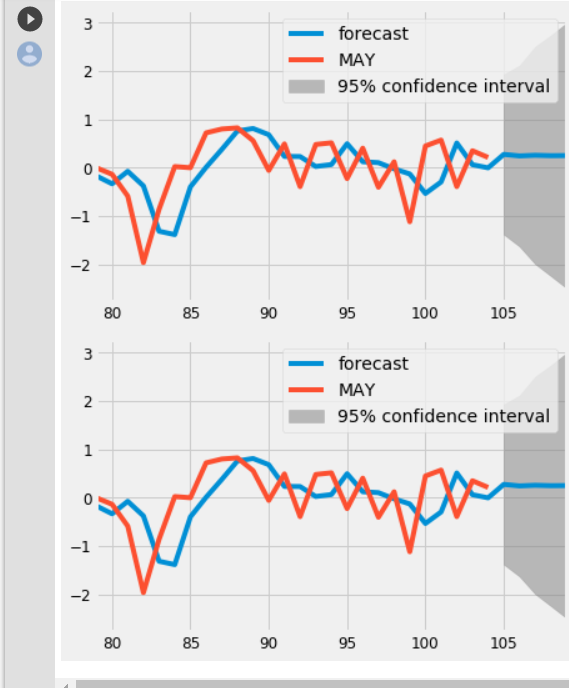
*106 rows × 1 columns*

**Code and subsequent output**

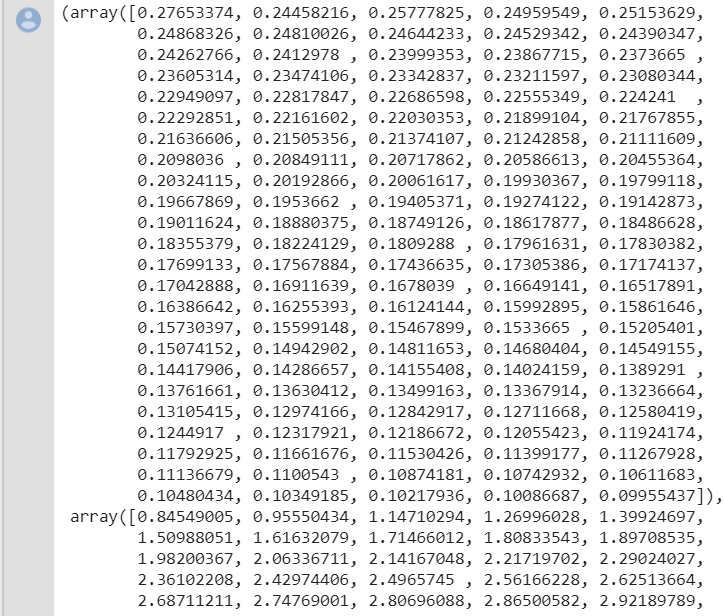
**#subsequent prediction within the 95% confidence range**

*results\_AR.plot\_predict(80,110)*

**Evaluation and Results**

**

*results\_AR.forecast(steps=120)*

**