Experiment No. :- 9

Student Name: Priyanshu Mathur **UID:** 20BEC1073
Branch: Electronics and Communication **Section/Group:** A
Semester: 7th **Date of Performance:** 03/11/2023
Subject Name: AIML **Subject Code:** 20ECA-445

1. **Aim of the practical:** Write a program to predict rainfall using Time Series Analysis.

2. **Tool Used:** Google Colab

3. **Theory:** Rainfall prediction is a crucial task in meteorology, agriculture, and water resource management. Making informed choices about agricultural planting, irrigation, and water conservation is crucial. Machine learning has developed into a potent tool for making highly accurate rainfall predictions. Rainfall prediction using machine learning involves the use of historical weather data and other relevant factors such as temperature, humidity, wind speed, and pressure to train a model that can accurately predict future rainfall.
   Autoregressive Integrated Moving Average (ARIMA) is a time series forecasting model that incorporates autocorrelation measures to model temporal structures within the time series data to predict future values. The auto regression part of the model measures the dependency of a particular sample with a few past observations. These differences are measured and integrated to make the data patterns stationary or minimize the obvious correlation with past data (since linear independence and no collinearity is one of the fundamental assumptions of the linear regression model).

4. **Steps for experiment/practical:**

Step 1: - Open Google Colab

Step 2: - Create a new notebook

Step 3: - Write the code given below and run it.
Program Code:-

```
 # Importing the libraries
import numpy as np import pandas as pd from matplotlib
import pyplot from statsmodels.tsa.arima_model import
ARIMA from statsmodels.tsa.stattools import acf,pacf
from statsmodels.tsa.stattools import adfuller from
statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

# Read and Pre-process Data data =
pd.read_csv("wet_freq_data.csv",sep="\t")
data.head()
# Converting our dataframe to time series
months = data.columns
years = data.Year Time
= [] for year in years:
for month in months:
if(month!='Year'):
        Time.append(str(year)+" "+month)
Time = pd.Index(Time)
# print Time
# print Time.shape

Values = [] for index, row in
data.iterrows():    r = []    r =
list(row)    r.pop(0)
Values.extend(r)

Values = pd.Index(Values)
# print Values #
print Values.shape
series = pd.DataFrame({'Time': Time, 'Values': Values})
dummy = series dummy.head()
series.index = pd.to_datetime(series.Time)
series.rename(columns={"Time":"Date"})
# series.drop(["Date"],axis=1)
series.head() # Visualize the Data
series_data = series.Values
minimum=series_data.min()
maximum=series_data.max()
series_data.plot() axes = pyplot.gca()
axes.set_ylim([minimum,maximum]) #setes ticks on y-axis according to min and max
```

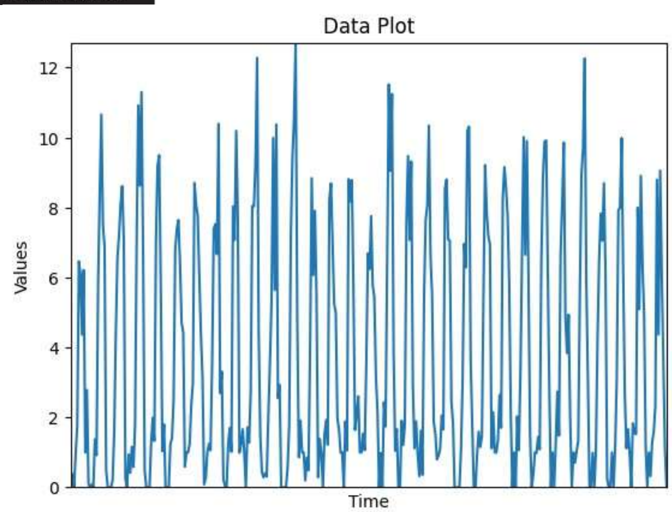
```
pyplot.tick_params(
    axis='x',          # changes apply to the x-axis
which='both',      # both major and minor ticks are affected
bottom=False,      # ticks along the bottom edge are off
top=False,         # ticks along the top edge are off
    labelbottom=False)

# pyplot.figure(15,5) pyplot.title("Data
Plot") pyplot.xlabel("Time")
pyplot.ylabel("Values")
pyplot.show()

X = series.Values
split = len(X) // 2  # Use integer division to get an integer index
X1, X2 = X[0:split], X[split:] mean1, mean2 = X1.mean(),
X2.mean() var1, var2 = X1.var(), X2.var()
print('mean1=%f, mean2=%f' % (mean1, mean2))
print('variance1=%f, variance2=%f' % (var1, var2)) yearly_mean =
series_data.rolling(window=12).mean() yearly_std =
series_data.rolling(window=12).std() orig =
pyplot.plot(series.Values,color='blue',label='Original') mean =
pyplot.plot(yearly_mean,color='red',label='Mean') std =
pyplot.plot(yearly_std,color='black',label='Standard Deviation')

pyplot.tick_params(
    axis='x',          # changes apply to the x-axis
which='both',      # both major and minor ticks are affected
bottom=False,      # ticks along the bottom edge are off
top=False,         # ticks along the top edge are off
    labelbottom=False)

pyplot.title("Data Plot") pyplot.xlabel("Time")
pyplot.ylabel("Values")

pyplot.legend(loc='best')
# Tests for Stationarity
# ADF (Augmented Dickey Fuller) Test
#dickey-fuller test def adf_test(timeseries):
#Perform Dickey-Fuller test:     print
('Results of Dickey-Fuller Test:')     dftest =
adfuller(timeseries, autolag='AIC')
```

```
             dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of
Observations Used'])     for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print (dfoutput)

adf_test(series_data)
# ACF Plot #extract
data values
zt = np.array(series_data)

#mean of data
mean = np.mean(zt)

#variance of data #var
= zt.var()
c0 = np.sum((zt - mean)*(zt - mean))/len(zt)

#calculate lag-wise auto-correlation
corr_coeffs=[] lags=[] for k in
range(len(zt)):
  series_1 = zt[k:]
series_2 = zt[:len(zt)-k]

  if len(series_1)!=len(series_2):
    print ("Error!!!")
else:
    num = np.sum((series_1 - mean)*(series_2 - mean))
den = c0*len(zt)
    coeff = num/den

    corr_coeffs.append(coeff)
lags.append(k)

#print the corr_coeffs and lags
# print "************************"
# print "correlation coeffs: ",corr_coeffs
# print "------------"
# print "lags: ",lags
# print "************************"
```

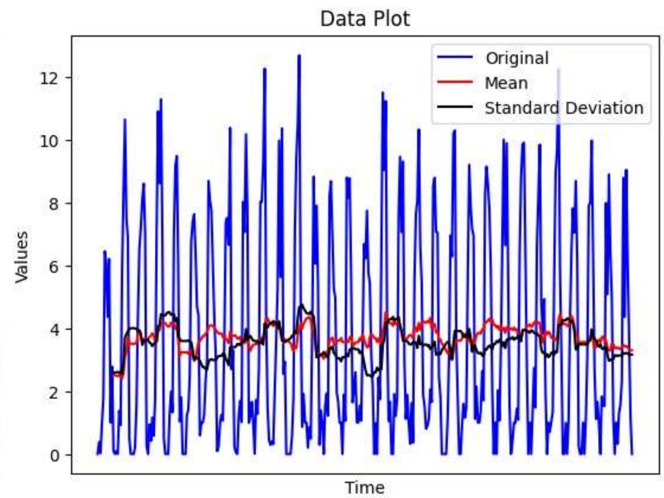
```
#plot autocorr vs lag
pyplot.title("calculated")
pyplot.bar(lags,corr_coeffs,width=0.2)
pyplot.axhline(0)
pyplot.show()

#plot the same using in-built func plot_acf(zt,lags=20)
pyplot.show()

#plot pacf using in-built function
plot_pacf(zt,lags=20) pyplot.show()
lag_acf = acf(series_data,nlags=20)
lag_pacf = pacf(series_data,nlags=20,method='ols')

pyplot.subplot(121) pyplot.plot(lag_acf)
pyplot.axhline(y=0,linestyle='--',color='gray') pyplot.axhline(y=-
1.96/np.sqrt(len(series_data)),linestyle='--',color='gray')
pyplot.axhline(y=1.96/np.sqrt(len(series_data)),linestyle='--',color='gray') pyplot.title("ACF")

pyplot.subplot(122) pyplot.plot(lag_pacf)
pyplot.axhline(y=0,linestyle='--',color='gray')
pyplot.axhline(y=-1.96/np.sqrt(len(series_data)),linestyle='--',color='gray')
pyplot.axhline(y=1.96/np.sqrt(len(series_data)),linestyle='--',color='gray')
pyplot.title("PACF")


# Fitting Model #
AR
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as pyplot

# Assuming series_data is your time series data

model = ARIMA(series_data, order=(2, 0, 0)) results_AR
= model.fit()

pyplot.plot(series_data, label='Original Series')
pyplot.plot(results_AR.fittedvalues, color='red', label='Fitted Values') pyplot.title('RSS: %.4f'
% sum((results_AR.fittedvalues - series_data)**2)) pyplot.legend()
pyplot.show()

# MA
```

```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Assuming series_data is your time series data

model = ARIMA(series_data, order=(0, 0, 4)) results_MA
= model.fit()

plt.plot(series_data, label='Original Series')
plt.plot(results_MA.fittedvalues, color='red', label='Fitted Values')
plt.title('RSS: %.4f' % sum((results_MA.fittedvalues - series_data)**2))
plt.legend() plt.show()
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Assuming series_data is your time series data

model = ARIMA(series_data, order=(3, 0, 2)) results_ARIMA
= model.fit()

plt.plot(series_data, label='Original Series')
plt.plot(results_ARIMA.fittedvalues, color='red', label='Fitted Values') plt.title('RSS:
%.4f' % sum((results_ARIMA.fittedvalues - series_data)**2)) plt.legend()
plt.show()
```

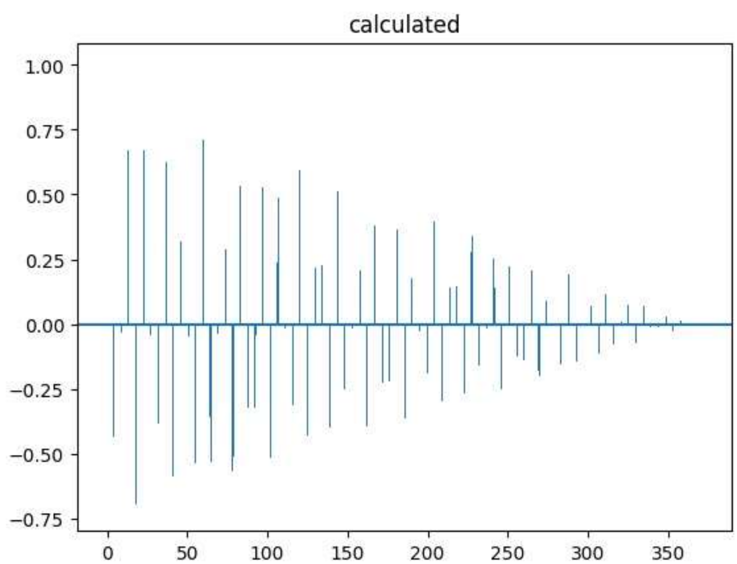
Plot :-



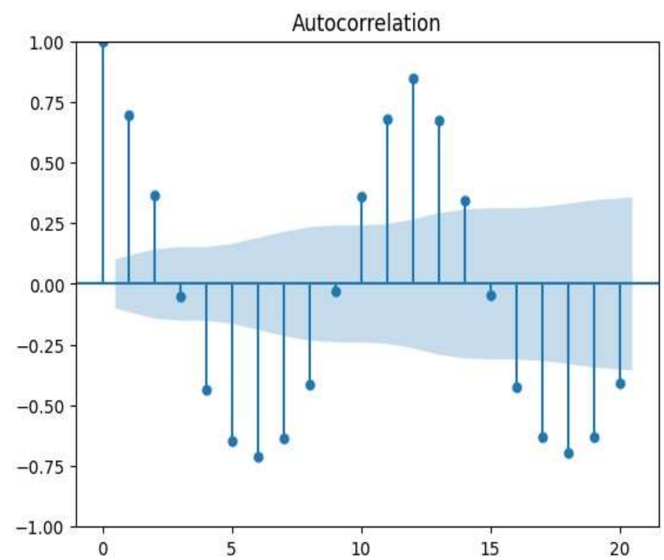
Data Plot
Values
Time



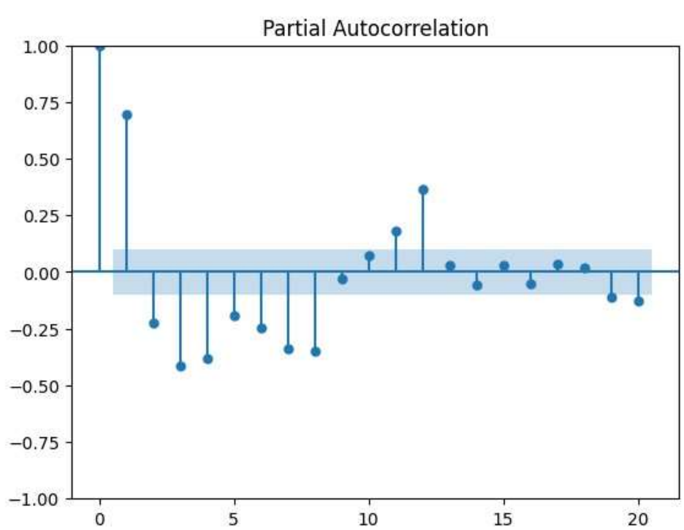
Data Plot
Original
Mean
Standard Deviation
Values
Time



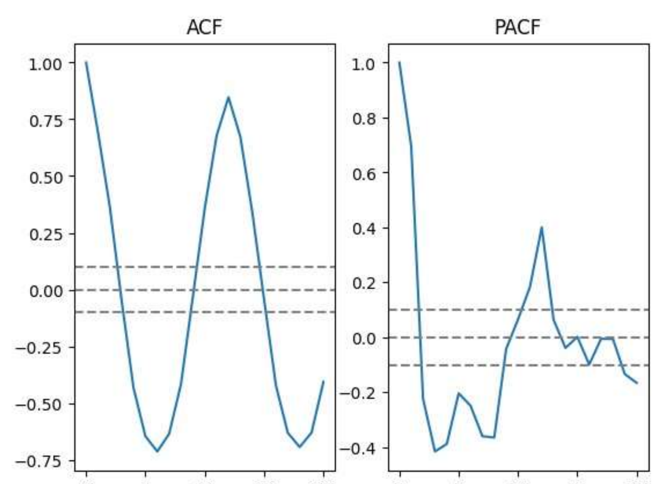
calculated



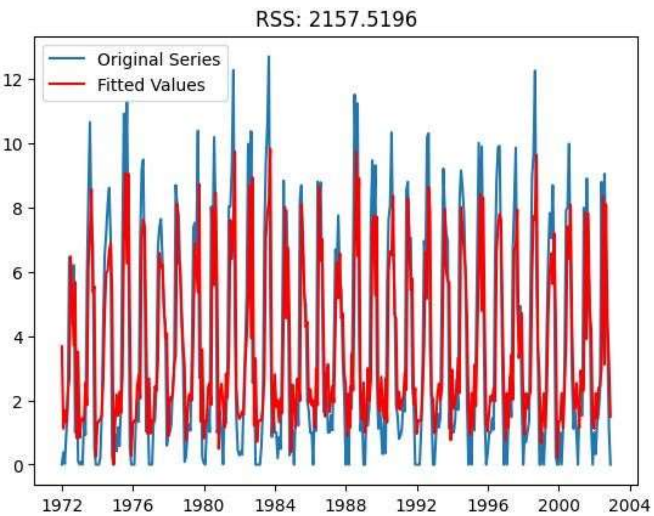
Autocorrelation

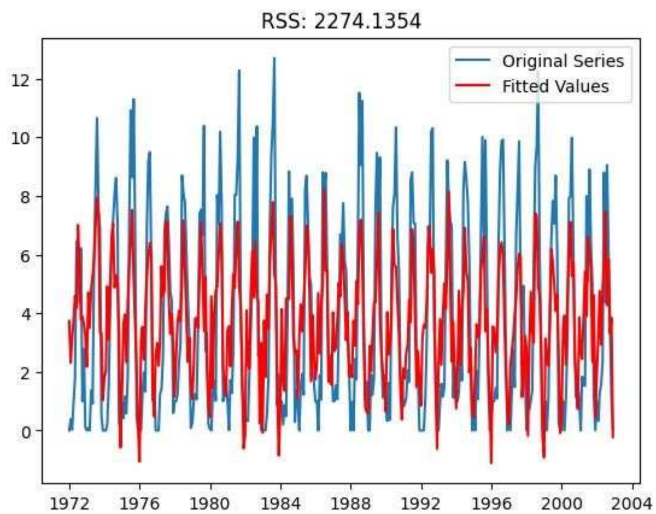


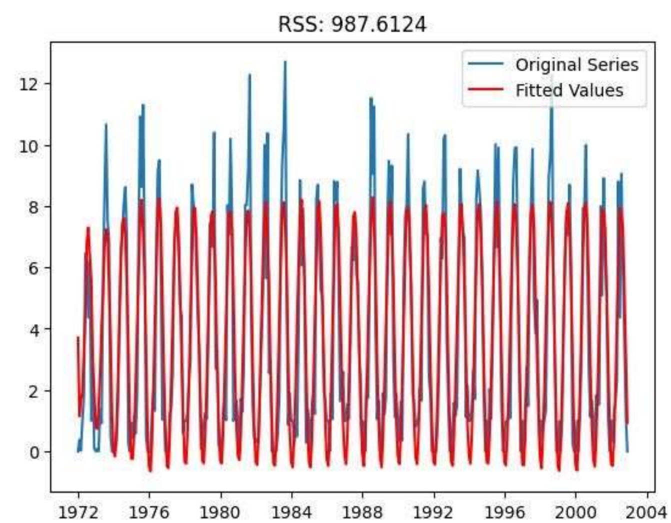
Partial Autocorrelation



ACF
PACF







Result and Discussion: - In this study, we have utilized time series analysis to forecast rainfall. We have used ARIMA Model for prediction. We have predicted future value by utilizing the past historical rainfall data. The ADF test assess the stationary of the time series data. This step is essential in preparing the data for the ARIMA model, ensuring that the assumptions of the model are met and enhancing the reliability of our predictions.

Learning outcomes (What I have learnt):

- Learnt about Time Series Analysis.
- Learnt about ARIMA model.
- Learnt about stationary test (ADF Test), ACF Test and PACF Test.