**Ex.no:8 Time Series Analysis**

**AIM:**

The aim of the program is to conduct time series analysis, forecasting, and visualize moving averages and autocorrelation.

**ALGORITHM:**

Step:1 Load and pre-process the dataset.

Step:2 Use the Augmented Dickey-Fuller (ADF) test to check for stationarity.

Step:3 Apply log transformation and differencing to make the series stationary.

Step:4 Plot the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF).

Step:5 Fit a Seasonal ARIMA (SARIMAX) model to the data.

Step:6 Forecast the gold prices until 2030.

Step:7 Plot the forecasted results and moving averages.

Step:8 Save the forecasted data and generate a table of predicted prices.

**CASE STUDY:**

This program analyses historical gold price data (in INR) to predict future prices up to the year 2030. The steps include loading and cleaning the data, checking for stationarity, and transforming the data to achieve stationarity. The program then fits a SARIMAX model to the transformed data, forecasts future prices, and visualizes the results along with moving averages. The predicted prices are saved and displayed in a tabular format for annual averages from 2024 to 2030.

**PROGRAM:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

import statsmodels.api as sm

from datetime import timedelta

# Load the dataset

df = pd.read\_csv("/content/data\_inr.csv")

df = df[['Name', 'Indian rupee']]

df['Name'] = pd.to\_datetime(df['Name'], format='%Y-%m-%d')

df.set\_index('Name', inplace=True)

df.dropna(inplace=True)

# Ensure all values are positive

df = df[df['Indian rupee'] > 0]

# Check stationarity using ADF test

result\_of\_adfuller = adfuller(df['Indian rupee'])

print('ADF Statistic: %f' % result\_of\_adfuller[0])

print('p-value: %f' % result\_of\_adfuller[1])

print('Critical Values:')

for key, value in result\_of\_adfuller[4].items():

print('\t%s: %.3f' % (key, value))

# Log transformation and differencing to make the series stationary

log\_transform = np.log(df['Indian rupee'])

diff\_log\_transform = log\_transform.diff().dropna()

# Check stationarity of the differenced log-transformed series

result\_of\_adfuller = adfuller(diff\_log\_transform)

print('ADF Statistic after log transformation: %f' % result\_of\_adfuller[0])

print('p-value: %f' % result\_of\_adfuller[1])

# Plot ACF and PACF

plt.figure()

plt.subplot(211)

plot\_acf(diff\_log\_transform, ax=plt.gca())

plt.subplot(212)

plot\_pacf(diff\_log\_transform, ax=plt.gca())

plt.show()

# Fit SARIMAX model

mod = sm.tsa.statespace.SARIMAX(df['Indian rupee'], order=(2, 1, 2), seasonal\_order=(2, 1, 2, 12),

enforce\_stationarity=False, enforce\_invertibility=False)

results = mod.fit()

# Forecasting until 2030

start\_date = df.index[-1]

end\_date = pd.to\_datetime('2030-12-31')

future\_dates = pd.date\_range(start=start\_date + pd.offsets.MonthBegin(), end=end\_date, freq='MS')

future\_steps = len(future\_dates)

# Extend the forecast to the desired future dates

future = results.get\_forecast(steps=future\_steps)

future\_df = future.conf\_int()

future\_df['forecast'] = future.predicted\_mean

future\_df.index = future\_dates

# Plot the forecast

plt.figure(figsize=(10, 6))

plt.plot(df.index, df['Indian rupee'], label='Observed')

plt.plot(future\_df.index, future\_df['forecast'], label='Forecast')

plt.fill\_between(future\_df.index, future\_df.iloc[:, 0], future\_df.iloc[:, 1], color='k', alpha=0.1)

plt.title('Gold Price Forecast (2024-2030)')

plt.xlabel('Year')

plt.ylabel('Price in INR')

plt.legend()

plt.show()

# Save the forecasted data

future\_df.to\_csv('gold\_price\_forecast\_2024\_2030.csv')

# Generate a table with years and predicted prices from 2024 to 2030

forecast\_table = future\_df['forecast'].resample('Y').mean()

forecast\_table.index = forecast\_table.index.year

forecast\_table = forecast\_table.reset\_index()

forecast\_table.columns = ['Year', 'Predicted Gold Price (INR)']

print(forecast\_table)

# Plot moving average

df['Moving Average (30 days)'] = df['Indian rupee'].rolling(window=30).mean()

df['Moving Average (90 days)'] = df['Indian rupee'].rolling(window=90).mean()

plt.figure(figsize=(10, 6))

plt.plot(df.index, df['Indian rupee'], label='Observed')

plt.plot(df.index, df['Moving Average (30 days)'], label='30-Day Moving Average')

plt.plot(df.index, df['Moving Average (90 days)'], label='90-Day Moving Average')

plt.title('Gold Price with Moving Averages')

plt.xlabel('Year')

plt.ylabel('Price in INR')

plt.legend()

plt.show()

# Plot autocorrelation and partial autocorrelation for the original series

plt.figure()

plt.subplot(211)

plot\_acf(df['Indian rupee'], ax=plt.gca())

plt.subplot(212)

plot\_pacf(df['Indian rupee'], ax=plt.gca())

plt.show()

# Plot autocorrelation and partial autocorrelation for the differenced log-transformed series

plt.figure()

plt.subplot(211)

plot\_acf(diff\_log\_transform, ax=plt.gca())

plt.subplot(212)

plot\_pacf(diff\_log\_transform, ax=plt.gca())

plt.show()

**OUTPUT:**

ADF Statistic: 0.314153

p-value: 0.977985

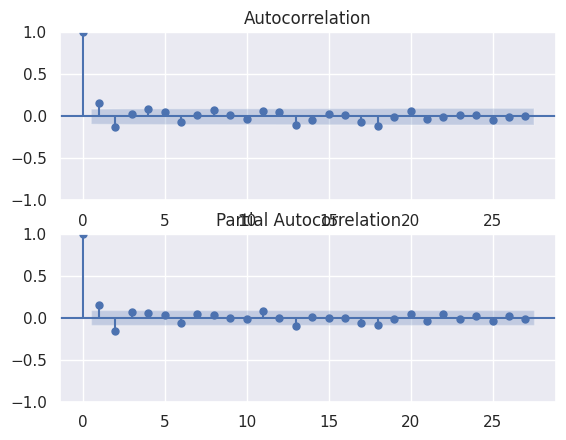
Critical Values:

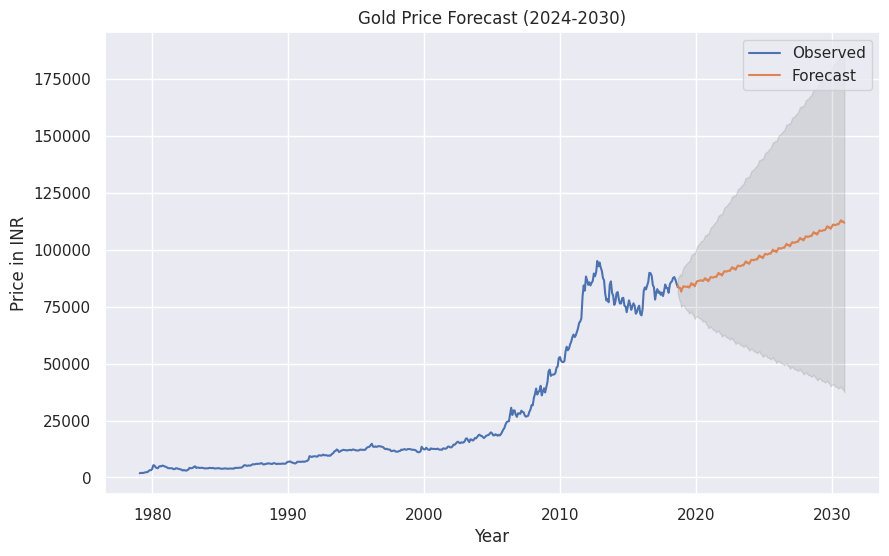
1%: -3.445

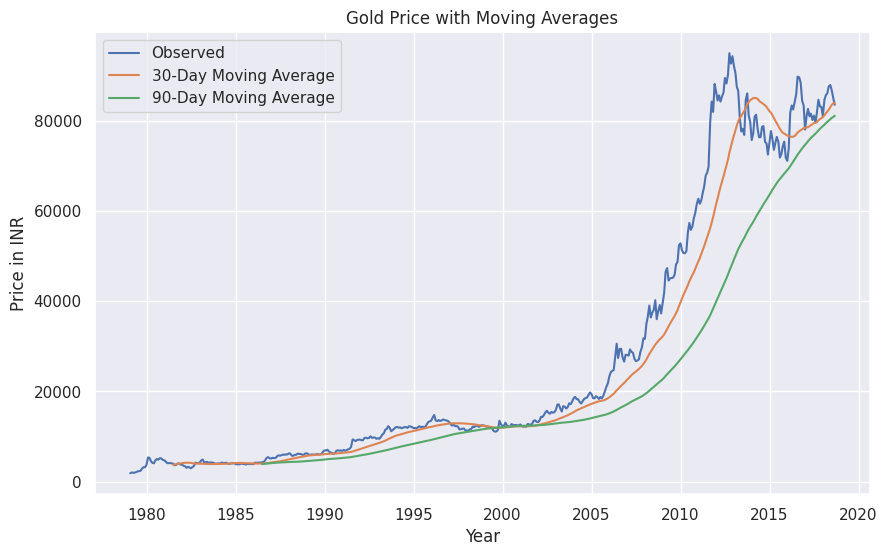
5%: -2.868

10%: -2.570

ADF Statistic after log transformation: -16.389687







Year Predicted Gold Price (INR)

0 2018 83132.620869

1 2019 84058.292859

2 2020 86483.562753

3 2021 88435.933317

4 2022 91016.922224

5 2023 93483.903941

6 2024 96067.557243

7 2025 98625.891138

8 2026 101206.228201

9 2027 103781.051766

10 2028 106360.013350

11 2029 108937.804982

12 2030 111516.389533

**RESULT:**

Thus, the python program to understand the time series analysis was executed successfully.