8. Create an ARIMA model for time series forecasting.

EX.N0 : 8	**Create an ARIMA model for time series forecasting.**
DATE : 07/04/2025	

AIM:

To Create an ARIMA model for time series forecasting.

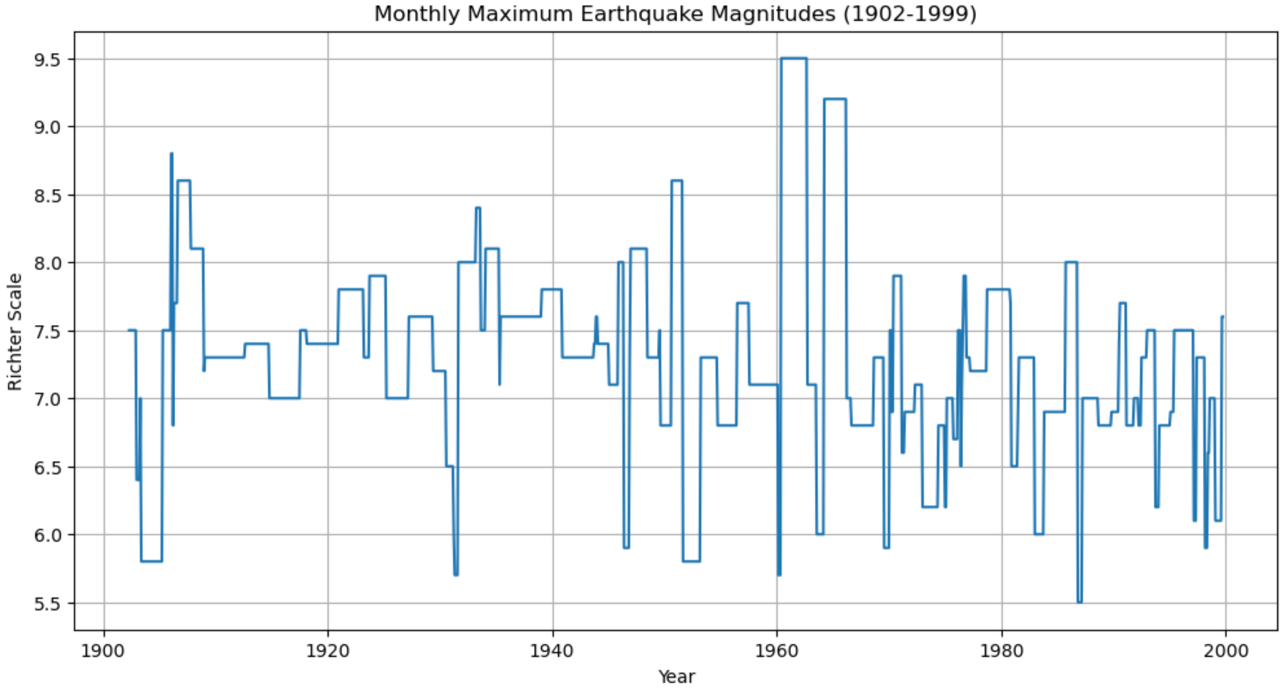
PROGRAM:

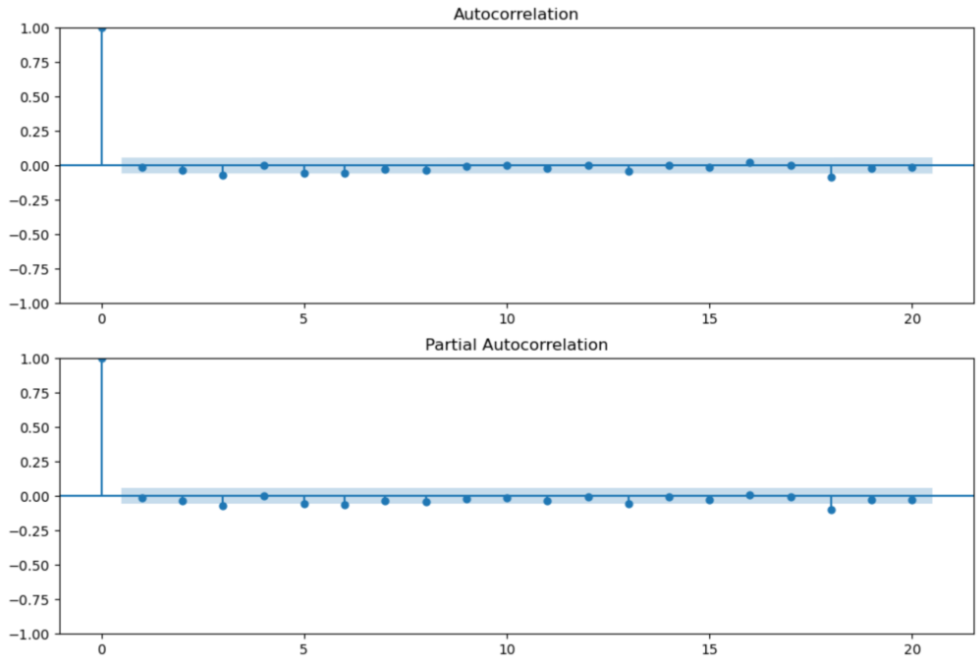
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
from math import sqrt
from datetime import datetime

# Load the data
df = pd.read_csv('earthquakes.csv')

# Convert month names to numbers
month_map = {
    'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6,
    'July': 7, 'August': 8, 'September': 9, 'October': 10, 'November': 11, 'December': 12
}
df['month'] = df['month'].map(month_map)

# Create datetime index
df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
df.set_index('date', inplace=True)
df.sort_index(inplace=True)

# Resample to monthly frequency (taking max magnitude per month)
ts = df['richter'].resample('M').max().ffill()  # Forward fill missing months

# Plot the original time series
plt.figure(figsize=(12, 6))
plt.plot(ts)
```

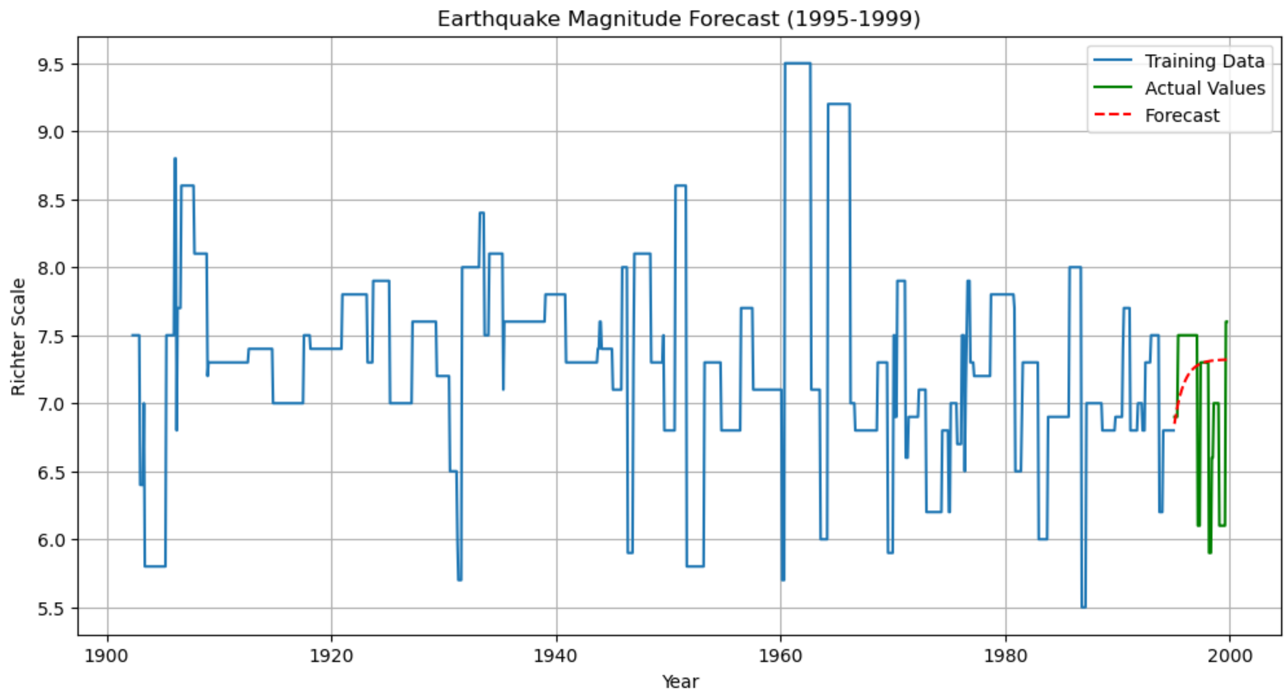
```
plt.title('Monthly Maximum Earthquake Magnitudes (1902-1999)')
plt.xlabel('Year')
plt.ylabel('Richter Scale')
plt.grid(True)
plt.show()

# Check for stationarity
def test_stationarity(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)

test_stationarity(ts)

# Differencing to make series stationary
ts_diff = ts.diff().dropna()
test_stationarity(ts_diff)

# Plot ACF and PACF
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))
plot_acf(ts_diff, lags=20, ax=ax1)
plot_pacf(ts_diff, lags=20, ax=ax2, method='ywm')
plt.show()

# Split data into train and test sets (last 5 years for testing)
train = ts.loc[:'1994-12-31']
test = ts.loc['1995-01-31':]

# Fit ARIMA model - parameters determined from ACF/PACF
model = ARIMA(train, order=(1, 1, 1))
model_fit = model.fit()
print(model_fit.summary())

# Forecast
forecast_steps = len(test)
forecast = model_fit.forecast(steps=forecast_steps)

# Plot forecasts
plt.figure(figsize=(12, 6))
plt.plot(train.index, train, label='Training Data')
plt.plot(test.index, test, label='Actual Values', color='green')
plt.plot(test.index, forecast, label='Forecast', color='red', linestyle='--')
```

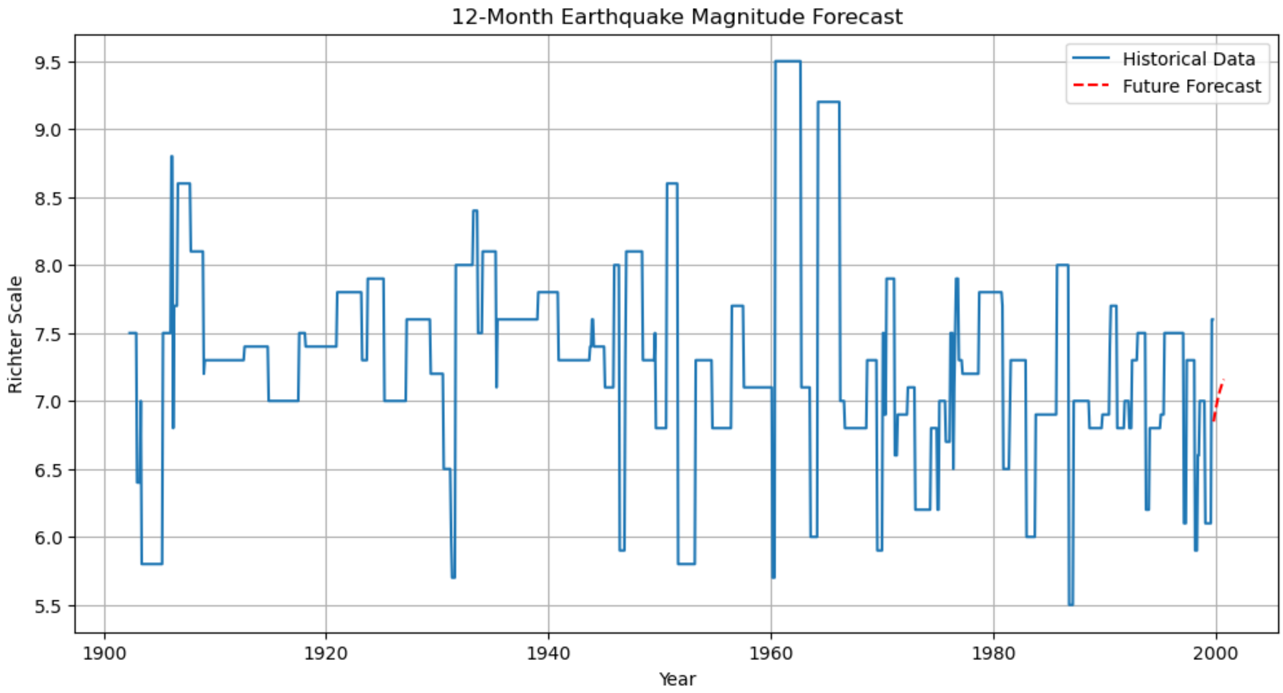
```
plt.title('Earthquake Magnitude Forecast (1995-1999)')
plt.xlabel('Year')
plt.ylabel('Richter Scale')
plt.legend()
plt.grid(True)
plt.show()

# Calculate RMSE
rmse = sqrt(mean_squared_error(test, forecast))
print(f'Test RMSE: {rmse:.3f}')

# Forecast future values (next 12 months)
future_forecast = model_fit.forecast(steps=12)
print("\n12-Month Future Forecast:")
print(future_forecast)

# Plot future forecast
future_dates = pd.date_range(start=ts.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts, label='Historical Data')
plt.plot(future_dates, future_forecast, label='Future Forecast', color='red', linestyle='--')
plt.title('12-Month Earthquake Magnitude Forecast')
plt.xlabel('Year')
plt.ylabel('Richter Scale')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:




Autocorrelation
Partial Autocorrelation


Earthquake Magnitude Forecast (1995-1999)
Training Data
Actual Values
Forecast
Richter Scale
Year



RESULT:

Thus, the program for Create an ARIMA model for time series forecasting is executed successfully.