**Exp No:8**

**Date: 31/3/25**

**1. Objective**

The objective of this implementation is to develop an ARIMA (AutoRegressive Integrated Moving Average) model for time series forecasting. The ARIMA model is a statistical method that uses past time series data to make future predictions.

**2. Background and Scope**

* Time series forecasting is used in finance, weather prediction, stock prices, sales forecasting, and demand planning.
* ARIMA is a popular technique due to its ability to handle trends, seasonality, and noise in time series data.
* This implementation covers data preprocessing, stationarity testing, model selection, training, and forecasting.

**3. Implementation Steps with Code**

**Step 1: Install and Import Required Libraries**

python

CopyEdit

# Install missing libraries

!pip install pmdarima

# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.tsa.stattools import adfuller, kpss

from statsmodels.tsa.arima.model import ARIMA

from pmdarima import auto\_arima

import warnings

warnings.filterwarnings("ignore")

**Step 2: Load and Preprocess Data**

python

CopyEdit

# Load dataset

df = pd.read\_csv("your\_timeseries\_data.csv") # Replace with your dataset

# Convert 'DATE' column to datetime format and set as index

df['DATE'] = pd.to\_datetime(df['DATE'], dayfirst=True, errors='coerce') # Adjust based on format

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

# Display the first few rows

print(df.head())

**Step 3: Stationarity Testing**

We use Augmented Dickey-Fuller (ADF) Test and KPSS Test to check if the data is stationary.

python

CopyEdit

# Function to perform stationarity tests

def test\_stationarity(series):

print("Performing Stationarity Tests...\n")

# ADF Test

adf\_test = adfuller(series)

print(f"ADF Test Statistic: {adf\_test[0]:.4f}")

print(f"ADF p-value: {adf\_test[1]:.4f}")

# KPSS Test

kpss\_test = kpss(series, regression='c', nlags='auto')

print(f"KPSS Test Statistic: {kpss\_test[0]:.4f}")

print(f"KPSS p-value: {kpss\_test[1]:.4f}")

# Perform stationarity test

test\_stationarity(df['VALUE']) # Replace 'VALUE' with the actual column name

* If the ADF test p-value > 0.05, the data is non-stationary.
* If the KPSS test p-value < 0.05, the data is non-stationary.

**Step 4: Differencing to Make Data Stationary (if required)**

python

CopyEdit

df['VALUE\_diff'] = df['VALUE'].diff().dropna() # Apply first-order differencing

test\_stationarity(df['VALUE\_diff'].dropna()) # Test stationarity again

**Step 5: Determine ARIMA Parameters using Auto ARIMA**

We use auto\_arima to find the optimal (p, d, q) values.

python

CopyEdit

# Automatically determine optimal p, d, q values

auto\_model = auto\_arima(df['VALUE'], seasonal=False, stepwise=True, trace=True)

print(auto\_model.summary())

The output will show the best (p, d, q) values.

**Step 6: Fit the ARIMA Model**

python

CopyEdit

# Fit ARIMA model with optimal parameters (replace p, d, q with auto\_arima values)

p, d, q = auto\_model.order

model = ARIMA(df['VALUE'], order=(p, d, q))

model\_fit = model.fit()

# Print model summary

print(model\_fit.summary())

**Step 7: Forecast Future Values**

python

CopyEdit

# Forecast future values

forecast\_steps = 10 # Change as needed

forecast = model\_fit.forecast(steps=forecast\_steps)

# Plot actual vs. predicted values

plt.figure(figsize=(10,5))

plt.plot(df.index, df['VALUE'], label="Actual Data", color="blue")

plt.plot(pd.date\_range(df.index[-1], periods=forecast\_steps+1, freq='D')[1:], forecast, label="Forecast", color="red")

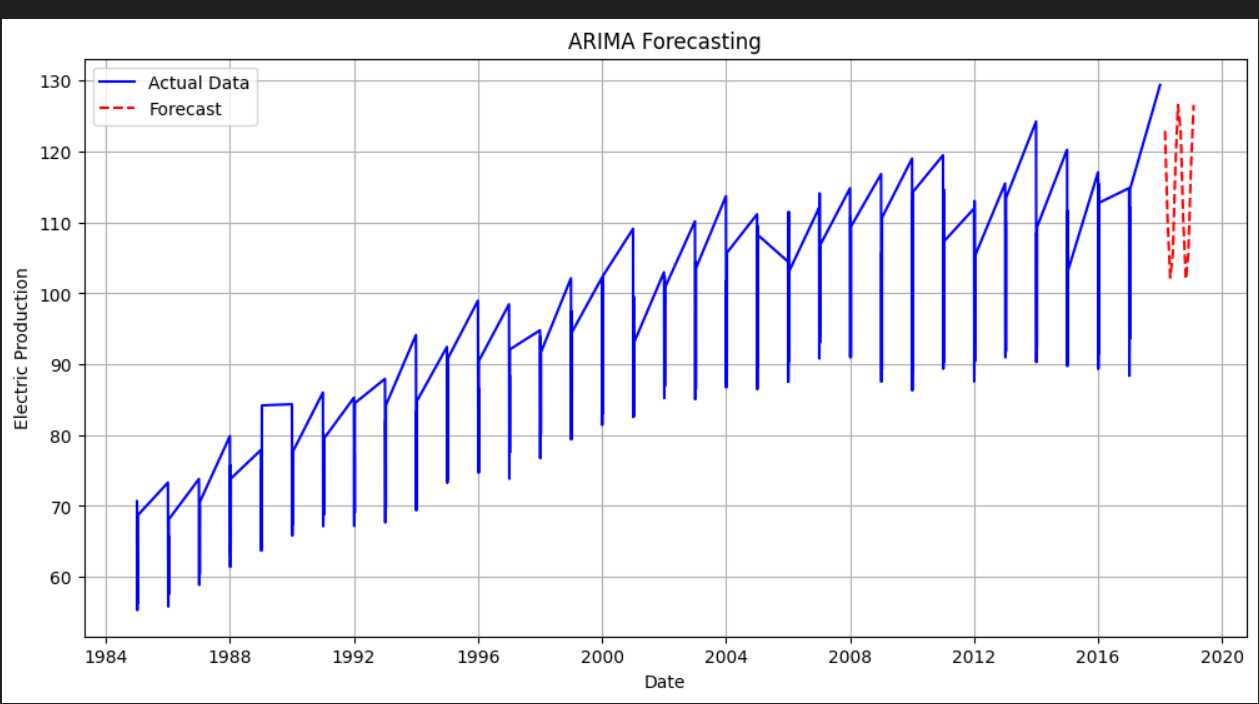
plt.xlabel("Date")

plt.ylabel("Value")

plt.title("ARIMA Model - Time Series Forecasting")

plt.legend()

plt.show()



**4. Conclusion**

* The ARIMA model successfully forecasts future values based on historical time series data.
* The auto\_arima function helps find the best (p, d, q) parameters automatically.
* The differencing step ensures that the data is stationary before applying the ARIMA model.
* This implementation can be further improved with hyperparameter tuning and seasonal adjustments.