EX:No.8 221501026

**Create an ARIMA model for time series forecasting.**

**Aim:**

Write a program to create an ARIMA model for time series forecasting.

**Algorithm:**

1. **Import necessary libraries**:  
Import numpy, pandas, matplotlib.pyplot, ARIMA from statsmodels.tsa.arima.model, and mean\_squared\_error from sklearn.metrics.

2. **Load the dataset**:  
Read the weather dataset using pandas.read\_csv() and parse the 'Date' column as datetime. Set the 'Date' column as the index.

3. **Select the target column**:  
Extract the 'Temperature' column from the DataFrame for time series forecasting.

4. **Split the data**:  
Calculate the training size as 80% of the total data.  
Split the data into training and testing sets using this calculated size.

5. **Define and fit the ARIMA model**:  
Initialize the ARIMA model with training data and set parameters (p=5, d=1, q=0).  
Fit the model using .fit().

6. **Forecast future values**:  
Forecast the temperature values for the length of the test set using .forecast().

7.**Visualize the training data**:  
Create a line plot for the training data using matplotlib.

8.**Visualize the actual test data**:  
Create a line plot for the test (actual) data.

9.**Visualize the forecasted vs actual test data**:  
Plot the ARIMA forecast along with the actual test values on the same graph to compare performance.

10.**Evaluate the model**:  
Calculate the Root Mean Squared Error (RMSE) between the forecast and actual test values using mean\_squared\_error().

11.**Print the RMSE value**.

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

# 🔄 Replace this with the correct relative or absolute path to your CSV file

file\_path = "coin\_crypto.csv"  # If it's in the same folder as your code

# Load the CSV file

df\_crypto = pd.read\_csv(file\_path)

# Convert 'Date' column to datetime and set as index

df\_crypto['Date'] = pd.to\_datetime(df\_crypto['Date'])

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

df\_crypto.sort\_index(inplace=True)

# Use 'Close' price for time series modeling

time\_series = df\_crypto['Close']

# Split data: 80% training, 20% testing

train\_size = int(len(time\_series) \* 0.8)

train, test = time\_series[:train\_size], time\_series[train\_size:]

# Fit ARIMA model

model = ARIMA(train, order=(5, 1, 0))

model\_fit = model.fit()

# Forecast

forecast = model\_fit.forecast(steps=len(test))

# Plot: Training Data

plt.figure(figsize=(10, 4))

plt.plot(train, label='Training Data', color='blue')

plt.title('Training Data (SOL Closing Price)')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot: Actual Test Data

plt.figure(figsize=(10, 4))

plt.plot(test, label='Actual Test Data', color='green')

plt.title('Actual Test Data (SOL Closing Price)')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot: Forecast vs Actual

plt.figure(figsize=(10, 4))

plt.plot(test.index, forecast, label='ARIMA Forecast', color='red')

plt.plot(test, label='Actual Test Data', color='green')

plt.title('ARIMA Forecast vs Actual (SOL Closing Price)')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Calculate and print RMSE

rmse = np.sqrt(mean\_squared\_error(test, forecast))

print(f'RMSE: {rmse:.2f}')

**Output:**

**A graph showing a line graph

AI-generated content may be incorrect.**

**A graph of a price

AI-generated content may be incorrect.**

**Result:**

Thus, the program to create an ARIMA model for time series forecasting was created successfully.