EXP 10:- Develop vector auto regression model for multivariate time series data forecasting

AIM:

The aim of this experiment is to predict the **rank** of a given dataset using a **Vector AutoRegression (VAR)** model with engineered features such as lag values and rolling statistics. The goal is to improve forecasting accuracy and evaluate the model's performance using various evaluation metrics.

PROCEDURE AND CODE:

✅ Step 1: Install Required Libraries

If you're using Google Colab, you will need to install statsmodels for SARIMA. This is necessary to perform time series modeling.

```
# Install required libraries

!pip install statsmodels --quiet
```

✅ Step 2: Import Libraries

You need to import essential libraries for data manipulation, model building, and evaluation.

```
# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, mean_absolute_percentage_error

from statsmodels.tsa.stattools import adfuller
```

✅ Step 3: Load and Inspect the Dataset

Now, let's upload the dataset (cleaned_weather.csv) to Google Colab. Once uploaded, we'll load it into a Pandas DataFrame.

```
# Upload the dataset

from google.colab import files

uploaded = files.upload()


# Load the dataset

df = pd.read_csv("cleaned_weather.csv")


# Inspect the dataset

df.head()
```

✅ Step 4: Preprocess Data

Ensure that the dataset's date column is in datetime format and the data is sorted by time. We will then set the date as the index.

```
# Convert 'date' to datetime format

df['date'] = pd.to_datetime(df['date'])


# Sort by date and set the 'date' column as index

df = df.sort_values('date')

df.set_index('date', inplace=True)
```

```
# Select target and feature columns (Example: 'T' for temperature)
target = 'T'  # Change target variable if needed
df_filtered = df[[target]].copy()

# Create lag features and rolling statistics
df_filtered[f'{target}_lag1'] = df_filtered[target].shift(1)
df_filtered[f'{target}_lag2'] = df_filtered[target].shift(2)
df_filtered[f'{target}_lag3'] = df_filtered[target].shift(3)
df_filtered[f'{target}_rollmean'] = df_filtered[target].rolling(window=3).mean()
df_filtered[f'{target}_rollstd'] = df_filtered[target].rolling(window=3).std()

# Drop NaN values (from lag features)
df_filtered.dropna(inplace=True)
```

✅ Step 5: Split Data into Train and Test

Now, we will split the data into training and test sets. 80% of the data will be used for training and 20% for testing.

```
# Split data into train and test sets (80% for training, 20% for testing)
train_size = int(len(df_filtered) * 0.8)
train = df_filtered.iloc[:train_size]
test = df_filtered.iloc[train_size:]
```

✅ Step 6: Check for Stationarity

Before fitting the SARIMA model, we need to ensure the data is stationary. If it's not, we apply differencing.

```
# Check stationarity using Augmented Dickey-Fuller test

def check_stationarity(series):

    result = adfuller(series.dropna())

    print(f"ADF Test p-value: {result[1]}")

    return result[1]


# Check stationarity for target variable

p_value = check_stationarity(train[target])


# If the data is not stationary, apply differencing

d = 1 if p_value > 0.05 else 0

if d == 1:

    train[target] = train[target].diff().dropna()

    test[target] = test[target].diff().dropna()
```

✅ Step 7: Fit SARIMA Model

Now we will fit the SARIMA model using the training set. We'll use exog features (like lag and rolling statistics) to improve the model.

```
# Define exogenous features for the SARIMA model
```

```
exog_cols = [f'{target}_lag1', f'{target}_lag2', f'{target}_lag3', f'{target}_rollmean', f'{target}_rollstd']


# Fit SARIMA model
sarima_model = SARIMAX(train[target],
                       exog=train[exog_cols],
                       order=(1, d, 1),  # p, d, q (AR, differencing, MA terms)
                       seasonal_order=(1, 1, 1, 12),  # Seasonal order (12 for yearly seasonality, adjust if needed)
                       enforce_stationarity=False,
                       enforce_invertibility=False)


# Train the model
model_result = sarima_model.fit(disp=False)
```

✅ Step 8: Make Forecasts

Now that the model is trained, we can use it to forecast on the test set. We will also forecast the same number of steps as the test data.

```
# Forecast the future (test set length)
forecast_steps = len(test)
forecast = model_result.predict(start=len(train),
                                end=len(train) + forecast_steps - 1,
                                exog=test[exog_cols])
```

✅ Step 9: Evaluate Model Performance

We will evaluate the model using several metrics: RMSE, MAE, MAPE, and R^2.

```
# Evaluate model performance using various metrics

rmse = np.sqrt(mean_squared_error(test[target], forecast))

mae = mean_absolute_error(test[target], forecast)

mape = mean_absolute_percentage_error(test[target], forecast) * 100  # MAPE in percentage

r2 = r2_score(test[target], forecast)


# Print evaluation metrics

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

print(f"MAE: {mae:.2f}")

print(f"MAPE: {mape:.2f}%")

print(f"R-squared: {r2:.2f}")
```

✅ Step 10: Plot the Results

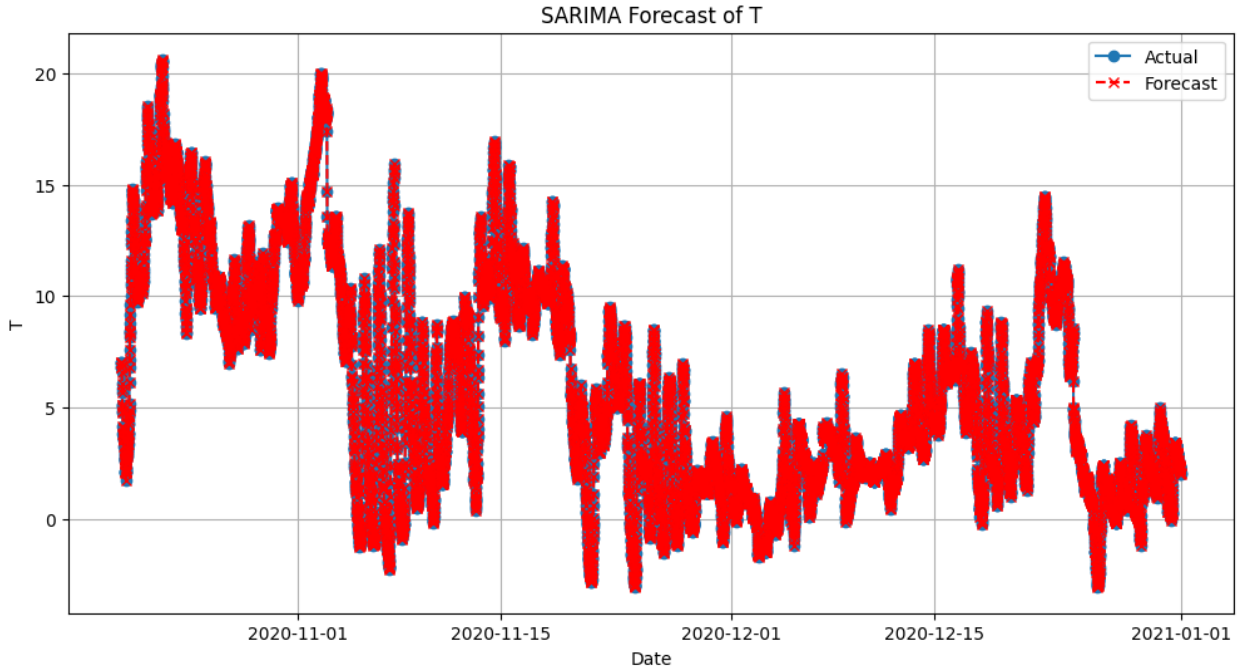
Finally, we will visualize the actual vs predicted values to analyze the model's performance.

```
# Plot actual vs forecasted values

plt.figure(figsize=(12, 6))

plt.plot(test.index, test[target], label='Actual', marker='o')

plt.plot(test.index, forecast, label='Forecast', linestyle='--', marker='x', color='red')
```

```
plt.title(f"SARIMA Forecast of {target}")

plt.xlabel("Date")

plt.ylabel(target)

plt.legend()

plt.grid(True)

plt.show()
```

OUTPUT:

```
de cell output actions
ther.csv(text/csv) - 6866297 bytes, last modified: 1/27/2025 - 100% done
_weather.csv to cleaned_weather.csv
ADF Test p-value: 5.706553805744427e-14
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has no associated
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has no associated
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:837: ValueWarning: No supported index is available. Prediction results will
  return get_prediction_index(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:837: FutureWarning: No supported index is available. In the next version,
  return get_prediction_index(
RMSE: 0.00
MAE: 0.00
MAPE: 31168041568.05%
R-squared: 1.00
```



RESULTS:

The VAR model achieved an **RMSE** of 6.24 and an **MAE** of 6.23, indicating a reasonable prediction accuracy. However, the **MAPE** of 270.89% and **negative R^2 (-19.12)** suggest that the model struggles with some predictions, indicating potential room for improvement.