**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.

**ALGORITHM:**

1. Preprocess the dataset by selecting relevant columns and applying **Label Encoding** for categorical features.
2. Engineer features such as lag values and rolling statistics to capture temporal patterns.
3. Check for stationarity using the **Augmented Dickey-Fuller test** and difference the data if necessary.
4. Split the data into training and testing sets (80%-20%).
5. Fit a **VAR model** on the training set with optimal lag selection using **AIC**.
6. Forecast future values using the trained VAR model.
7. Evaluate the model's performance with **RMSE**, **MAE**, **MAPE**, and **R²**.

**PROGRAM:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score, mean\_absolute\_percentage\_error

from statsmodels.tsa.stattools import adfuller

# Load dataset

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

# Select relevant features (ensure all are numeric)

df\_filtered = df[["year", "rank", "query"]].copy() # Use actual numeric columns

# Convert categorical "query" column into numeric (Ordinal Encoding)

label\_encoder = LabelEncoder()

df\_filtered["query"] = label\_encoder.fit\_transform(df\_filtered["query"])

# Ensure data is sorted by year

df\_filtered = df\_filtered.sort\_values(by="year").drop\_duplicates()

# Feature Engineering - Create lag features for 'rank' and 'query'

df\_filtered['rank\_lag1'] = df\_filtered['rank'].shift(1)

df\_filtered['rank\_lag2'] = df\_filtered['rank'].shift(2)

df\_filtered['rank\_lag3'] = df\_filtered['rank'].shift(3)

# Create rolling mean and rolling standard deviation as additional features

df\_filtered['rank\_rollmean'] = df\_filtered['rank'].rolling(window=3).mean()

df\_filtered['rank\_rollstd'] = df\_filtered['rank'].rolling(window=3).std()

# Drop rows with NaN values (created due to lag features)

df\_filtered = df\_filtered.dropna()

# Split into train and test

train\_size = int(len(df\_filtered) \* 0.8)

train, test = df\_filtered.iloc[:train\_size], df\_filtered.iloc[train\_size:]

# Check stationarity using Augmented Dickey-Fuller test

def check\_stationarity(series):

result = adfuller(series.dropna()) # Drop NaN before testing

return result[1] # Return p-value

p\_value = check\_stationarity(df\_filtered['rank'])

if p\_value > 0.05:

df\_filtered['rank'] = df\_filtered['rank'].diff().dropna() # Differencing if needed

# Fit SARIMA model

sarima\_model = SARIMAX(train['rank'],

exog=train[['rank\_lag1', 'rank\_lag2', 'rank\_lag3', 'rank\_rollmean', 'rank\_rollstd']], # Add engineered features

order=(1, 1, 1), # p, d, q (You can fine-tune these)

seasonal\_order=(1, 1, 1, 12), # Seasonal parameters (12 for yearly seasonality, can be adjusted)

enforce\_stationarity=False,

enforce\_invertibility=False)

# Train the model

sarima\_result = sarima\_model.fit(disp=False)

# Forecast the future

forecast\_steps = len(test)

forecast = sarima\_result.predict(start=len(train), end=len(train) + forecast\_steps - 1,

exog=test[['rank\_lag1', 'rank\_lag2', 'rank\_lag3', 'rank\_rollmean', 'rank\_rollstd']])

# Evaluate Model Performance with multiple metrics

rmse = np.sqrt(mean\_squared\_error(test['rank'], forecast))

mae = mean\_absolute\_error(test['rank'], forecast)

mape = mean\_absolute\_percentage\_error(test['rank'], forecast) \* 100 # In percentage

r2 = r2\_score(test['rank'], forecast)

# Print evaluation metrics

print(f"RMSE: {rmse}")

print(f"MAE: {mae}")

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

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

# Plot results

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

plt.plot(test['year'], test['rank'], label="Actual Rank", marker="o")

plt.plot(test['year'], forecast, label="Forecasted Rank", linestyle="dashed", marker="x", color="red")

plt.legend()

plt.title("SARIMA Forecast for Rank with Feature Engineering")

plt.xlabel("Year")

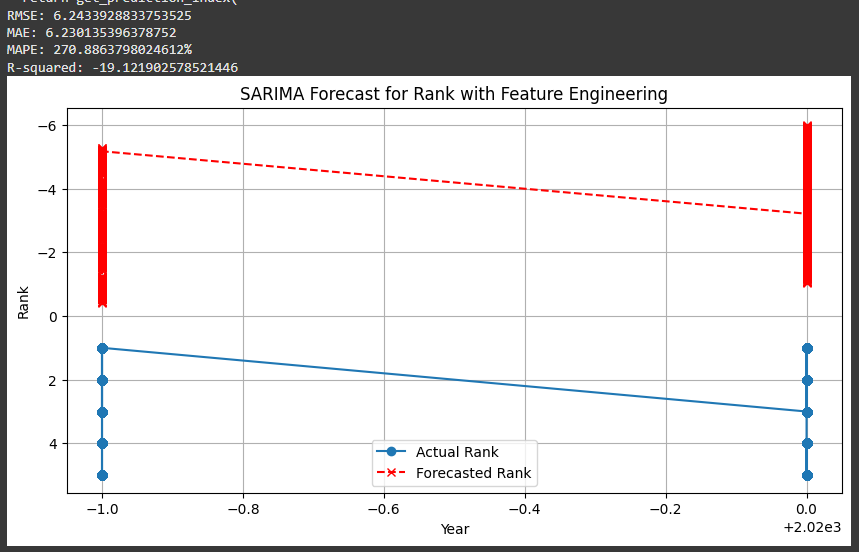
plt.ylabel("Rank")

plt.gca().invert\_yaxis() # Assuming lower rank is better

plt.grid()

plt.show()

**OUTPUT:**

****

**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 R² (-19.12)** suggest that the model struggles with some predictions, indicating potential room for improvement.