10. Develop vector auto regression model for multivariate time series data forecasting.

AIM:

To develop a **Vector Auto Regression (VAR)** model for forecasting multiple interrelated time series variables (e.g., different expense categories) using historical data.

PROCEDURE:

1) Import Libraries
2) Load the Dataset
3) Preprocess Data (parse dates, handle missing values)
4) Visualize Time Series (optional)
5) Check for Stationarity (ADF Test)
6) Make Series Stationary (if needed via differencing)
7) Train-Test Split
8) Fit the VAR Model
9) Forecast and Inverse Differencing
10) Evaluate and Plot Predictions

CODE:

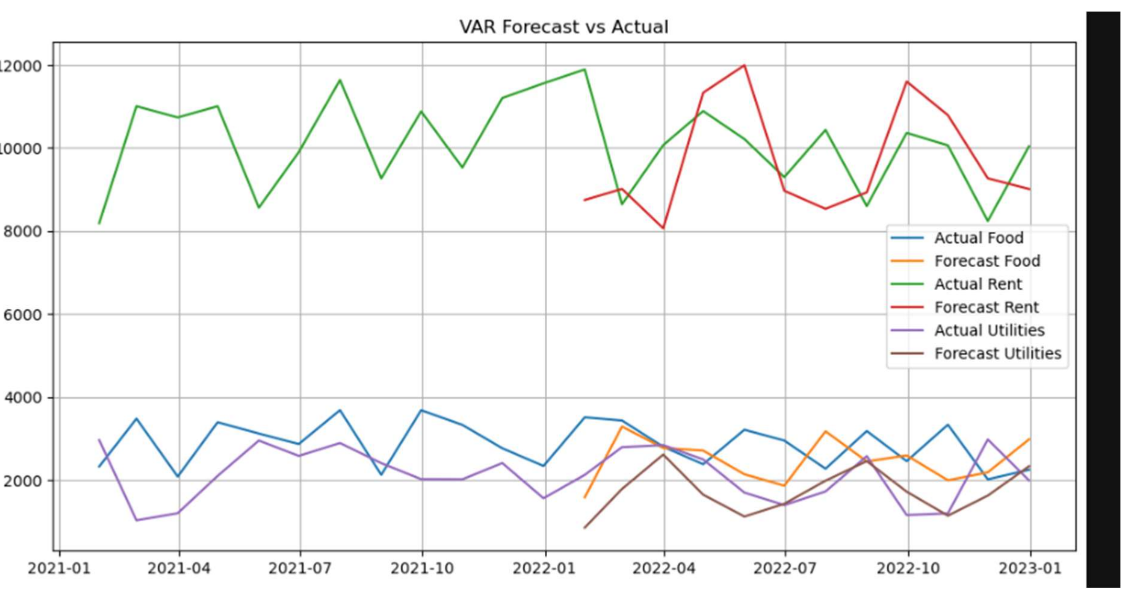
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.api import VAR

from statsmodels.tsa.stattools import adfuller


# 1. Load Data

df = pd.read_csv('expense_data_1.csv')


# 2. Parse date column and set as index

if 'date' in df.columns:

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

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

```
elif 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df.set_index('Date', inplace=True)

# 3. Select numeric columns
data = df.select_dtypes(include='number').dropna()
print("Using numeric columns:", list(data.columns))

# 4. Check stationarity and difference if needed
def adf_test(series, name):
    result = adfuller(series.dropna())
    print(f'ADF Test for {name}: p={result[1]:.4f} - {"Stationary" if result[1] < 0.05 else "Non-
Stationary"}')

print("\nADF Test Results:")
print("\nADF Test Results:")
for col in data.columns:
    series = data[col].dropna()
    if series.empty:
        print(f"Skipping ADF Test for {col}: Series is empty.")
        continue
    if series.nunique() <= 1:
        print(f"Skipping ADF Test for {col}: Constant or nearly constant values.")
        continue
    try:
        result = adfuller(series)
        print(f'ADF Test for {col}: p={result[1]:.4f} - {"Stationary" if result[1] < 0.05 else "Non-
Stationary"}')
    except Exception as e:
        print(f"ADF Test for {col} failed: {e}")
```

```python
# 5. Differencing to make stationary
data_diff = data.diff().dropna()

# 6. Split into training and test sets
n_obs = 10  # Number of time steps to forecast
train, test = data_diff[:-n_obs], data_diff[-n_obs:]

# 7. Train VAR model
model = VAR(train)
# Automatically determine max lags based on data size
max_lags_allowed = min(5, int(len(train) / 5))  # Rule of thumb: lag << number of rows
print(f"Using maxlags = {max_lags_allowed}")

model_fitted = model.fit(maxlags=max_lags_allowed, ic='aic')

print(model_fitted.summary())

# 8. Forecast
lag_order = model_fitted.k_ar
forecast_input = train.values[-lag_order:]
forecast = model_fitted.forecast(y=forecast_input, steps=n_obs)
forecast_df = pd.DataFrame(forecast, index=test.index, columns=data.columns)

# 9. Inverse Differencing
def invert_transformation(train_data, forecast_data):
    forecast = forecast_data.copy()
    for col in train_data.columns:
```

```
        forecast[col] = train_data[col].iloc[-1] + forecast[col].cumsum()
    return forecast

forecast_final = invert_transformation(data, forecast_df)

# 10. Plot Actual vs Forecast
for col in data.columns:
    plt.figure(figsize=(10, 4))
    plt.plot(data[col][-n_obs:], label='Actual')
    plt.plot(forecast_final[col], label='Forecast')
    plt.title(f'{col} - Actual vs Forecast')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

OUTPUT:



RESULT:

The program To develop a **Vector Auto Regression (VAR)** model for forecasting multiple interrelated time series variables (e.g., different expense categories) using historical data is implemented successfully.