5.Develop a linear regression model for forecasting time series data

**Aim:**

To develop a linear regression model that forecasts future values of a given time series dataset.

**Procedure:**

1. **Import Required Libraries**: We need libraries like Pandas for data handling, Matplotlib for visualization, and Scikit-learn for building the linear regression model.
2. **Load Data**: Import the time series data.
3. **Preprocess Data**: Handle any missing values, and transform the time series into a suitable format (for example, creating time-based features).
4. **Feature Engineering**: Generate the required features for the linear regression model. For time series, we will typically create lag features (previous time steps) as predictors.
5. **Split Data into Training and Test Sets**: Divide the data into a training set and a test set (usually an 80-20 split).
6. **Build the Model**: Use Scikit-learn's LinearRegression to fit the model on the training set.
7. **Evaluate the Model**: Use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared to evaluate the model performance.
8. **Forecasting**: Use the model to make predictions on future time points.
9. **Visualize the Results**: Plot the original and predicted values to visually inspect the performance of the model.

**Code:**

Below is a Python implementation using the above procedure:

python

CopyEdit

# Step 1: Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

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

# Step 2: Load the data (Here, we simulate time series data as an example)

# In practice, you will replace this with actual data loading

# For example, you can use pd.read\_csv('your\_data.csv') for a CSV file.

data = pd.Series([100, 110, 115, 120, 130, 125, 135, 140, 150, 155, 160, 170, 175, 180, 185])

# Step 3: Preprocess the data

# Create lag features for time series

def create\_lag\_features(data, lag=1):

lagged\_data = pd.DataFrame(data)

for i in range(1, lag+1):

lagged\_data[f'lag\_{i}'] = lagged\_data['0'].shift(i)

lagged\_data.dropna(inplace=True)

return lagged\_data

# Here we use lag=1 for simplicity

lagged\_data = create\_lag\_features(data, lag=1)

# Step 4: Feature Engineering

X = lagged\_data.drop(columns='0') # Features (lags)

y = lagged\_data['0'] # Target variable (current value)

# Step 5: Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Step 6: Build the Linear Regression Model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Step 7: Evaluate the Model

y\_pred = model.predict(X\_test)

# Calculate evaluation metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Display the evaluation metrics

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Mean Squared Error (MSE): {mse}')

print(f'R-squared (R2): {r2}')

# Step 8: Forecasting

future\_lag = np.array([y[-1]]) # Using the last known value as a "lag" for future prediction

future\_pred = model.predict(future\_lag.reshape(1, -1))

# Print future forecast

print(f'Forecast for the next time point: {future\_pred[0]}')

# Step 9: Visualize the results

plt.figure(figsize=(10, 6))

plt.plot(data, label='Actual')

plt.plot(range(len(X\_train), len(X\_train) + len(X\_test)), y\_pred, label='Predicted', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.show()

**Explanation of the Code:**

* **Step 1**: We import the necessary libraries for handling data, building the model, and visualizing the results.
* **Step 2**: In this example, we simulate a simple time series data. In a real-world scenario, you would load your dataset here.
* **Step 3**: A function create\_lag\_features() is defined to create lag features. For example, lag\_1 will hold the value from the previous time step.
* **Step 4**: We separate the features (lagged values) and the target variable (current values).
* **Step 5**: We split the dataset into training and test sets. The train\_test\_split function is used with shuffle=False to preserve the time order.
* **Step 6**: We build the linear regression model using LinearRegression from Scikit-learn and train it on the training set.
* **Step 7**: The model is used to make predictions on the test set. We then compute common metrics like MAE, MSE, and R-squared to evaluate its performance.
* **Step 8**: The model is used to forecast future values. We use the last value in the time series as a lag to make predictions.
* **Step 9**: We visualize the actual and predicted values using Matplotlib.

**Result:**

After running the code, the output will display the evaluation metrics and the forecast for the next time point.