Experiment 3: Linear Regression Model for Forecasting Time Series Data

1. **Importing Necessary Libraries**

**Explanation:**

This section imports the necessary libraries for data handling, numerical operations, plotting, and machine learning. These libraries are essential for reading the dataset, preprocessing data, building the model, and evaluating performance.

**Corresponding Code:**

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, TimeSeriesSplit from sklearn.linear\_model import LinearRegression

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

1. **Loading the Dataset**

**Explanation:**

The dataset is loaded from a CSV file into a pandas DataFrame. The dataset used contains COVID-19 case data over time. This data will be used to forecast the number of confirmed cases.

**Corresponding Code:**

file\_path = "/content/time-series-19-covid-combined (1).csv" df = pd.read\_csv(file\_path)

1. **Data Preprocessing**

**Explanation:**

* + Convert the **Date** column to datetime format for proper time-series analysis.
  + Sort the dataset by date to maintain the chronological order.
  + Group the data by date to get the **total number of confirmed, deaths, and recovered cases** globally per day.
  + Perform a **log transformation** on the confirmed cases to make exponential growth trends easier to model.
  + Generate **lag features**, where each day's data depends on data from the past 7 days (weekly lags).

**Corresponding Code:**

df['Date'] = pd.to\_datetime(df['Date']) df = df.sort\_values(by='Date')

df\_grouped = df.groupby('Date')[['Confirmed', 'Deaths', 'Recovered']].sum().reset\_index()

df\_grouped['Log\_Confirmed'] = np.log1p(df\_grouped['Confirmed'])

num\_lags = 7

for i in range(1, num\_lags + 1):

df\_grouped[f'Confirmed\_lag\_{i}'] = df\_grouped['Confirmed'].shift(i) df\_grouped[f'Deaths\_lag\_{i}'] = df\_grouped['Deaths'].shift(i) df\_grouped[f'Recovered\_lag\_{i}'] = df\_grouped['Recovered'].shift(i)

df\_grouped.dropna(inplace=True)

1. **Feature Selection and Target Variable**

**Explanation:**

* + The features consist of **lagged confirmed, deaths, and recovered cases**

from the past 7 days.

* + The target variable is the **log-transformed confirmed cases**, helping the model capture exponential trends.

**Corresponding Code:**

features = [f'Confirmed\_lag\_{i}' for i in range(1, num\_lags + 1)] + \ [f'Deaths\_lag\_{i}' for i in range(1, num\_lags + 1)] + \ [f'Recovered\_lag\_{i}' for i in range(1, num\_lags + 1)]

target = 'Log\_Confirmed'

1. **Splitting the Data into Training and Testing Sets**

**Explanation:**

* + The data is split using **TimeSeriesSplit**, which ensures that training data always precedes testing data.
  + This is important in time series data to avoid data leakage (future data influencing past predictions).

**Corresponding Code:**

X = df\_grouped[features] y = df\_grouped[target]

tscv = TimeSeriesSplit(n\_splits=5)

1. **Training the Linear Regression Model**

**Explanation:**

* + A **Linear Regression** model is created.
  + The model is trained and validated on each fold generated by TimeSeriesSplit.
  + This approach mimics real-time forecasting, training only on past data and validating on future data.

**Corresponding Code:**

model = LinearRegression()

for train\_index, test\_index in tscv.split(X):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index] y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

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

1. **Making Predictions**

**Explanation:**

* + After training on each fold, predictions are made for the test set.
  + Predictions are made in the **log scale**.

**Corresponding Code:**

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

1. **Model Evaluation**

**Explanation**

* + **Mean Absolute Error (MAE)**: Measures average prediction error.
  + **Mean Squared Error (MSE)**: Measures average squared error.
  + **R² Score**: Measures how much variance is explained by the model.

**Corresponding Code:**

mae = mean\_absolute\_error(y\_test, y\_pred) mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}") print(f"Mean Squared Error (MSE): {mse:.2f}") print(f"R² Score: {r2:.4f}")

1. **Visualization of Actual vs Predicted Values**

**Explanation:**

* + After training on the full dataset, the final predictions are **inverse- transformed** (exponentiated) back to the actual case counts.
  + A plot is generated showing **actual vs predicted cases** over time.

**Corresponding Code:**

X\_final\_test = X.iloc[test\_index] y\_final\_test = y.iloc[test\_index] y\_final\_pred = model.predict(X\_final\_test)

y\_final\_pred\_actual = np.expm1(y\_final\_pred) y\_final\_test\_actual = np.expm1(y\_final\_test)

df\_results = pd.DataFrame({'Date': df\_grouped['Date'].iloc[-len(y\_final\_test):], 'Actual Confirmed': y\_final\_test\_actual,

'Predicted Confirmed': y\_final\_pred\_actual}) print(df\_results.head())

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

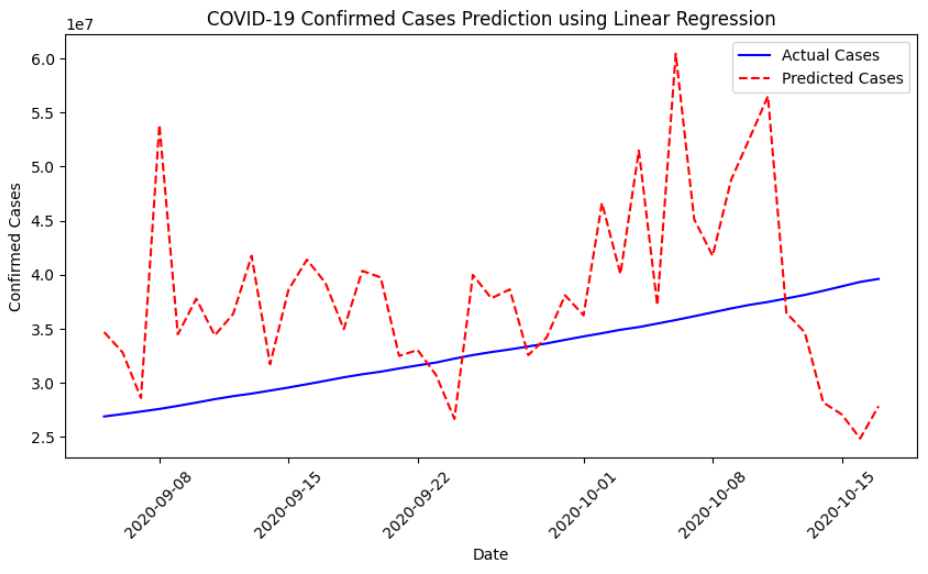
plt.plot(df\_results['Date'], df\_results['Actual Confirmed'], label='Actual Cases', color='blue')

plt.plot(df\_results['Date'], df\_results['Predicted Confirmed'], label='Predicted Cases', color='red', linestyle='dashed')

plt.xlabel('Date') plt.ylabel('Confirmed Cases') plt.legend()

plt.title('COVID-19 Confirmed Cases Prediction using Linear Regression') plt.xticks(rotation=45)

plt.show()



**Result**

Thus, the linear regression model for forecasting time series data (COVID-19 confirmed cases) has been successfully developed and evaluated.