**Exp no: 1 Implement programs for time series data cleaning, loading and handling times series data and pre-processing techniques.**

**Date: 24/1/25**

**Objectives:**

The objective of this preprocessing pipeline for a time series sales dataset is to prepare the data for predictive modeling. The steps aim to handle missing values, remove anomalies, and ensure proper time-based ordering. The goal is to process the data so that models can accurately predict future sales trends, detect seasonality patterns, and generate forecasts. By addressing issues such as missing timestamps and ensuring the data is in a consistent format, we improve the quality of input for time series forecasting models.

**Background-Scope:**

In time series data, each observation is indexed by time, often involving sales data over days, months, or years. Missing values, outliers, and improper time ordering can introduce bias into the analysis. This preprocessing scope focuses on handling these issues by filling gaps, removing outliers, and ensuring that the data is correctly indexed. Additionally, transforming features like seasonal components and trend patterns can help improve forecast accuracy. Proper preprocessing lays the groundwork for time series forecasting models to detect trends, predict future sales, and guide business decisions.

**Steps for Time Series Sales Data Preprocessing:**

**Step 1: Load the Dataset**

* **Load** the sales dataset from a CSV file into a Pandas DataFrame.
* **Check** for missing values in the dataset and identify any columns with null values.

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

# Step 1: Load the dataset

# Assuming 'sales.csv' has columns 'Order Date' and 'Sales'

df = pd.read\_csv('sales.csv')

**Step 2: Visualize Missing Values**

* **Use missingno** to visualize missing values in the dataset.
* This visualization helps identify patterns or areas where missing values are concentrated.

print("\nMissing Values:")

print(df.isnull().sum())

**Step 3: Handle Missing or Invalid Values**

* **Replace invalid values** (like zeros in numerical columns) with None (NaN), as zero values might be considered invalid in some cases.
* **Fill missing values** in numeric columns using the **median** of each column to ensure that missing data does not distort the dataset.

# Handle missing values (if any) using forward fill

df.fillna(method='ffill', inplace=True)

# Convert 'Order Date' column to datetime format

df['Order Date'] = pd.to\_datetime(df['Order Date'], format='%d/%m/%Y')

# Set 'Order Date' as the index

df.set\_index('Order Date', inplace=True)

# Display the cleaned dataset

print("\nCleaned Dataset:")

print(df.head())

**Step 4: Remove Duplicate Rows**

* **Check** for duplicate rows in the dataset.
* **Remove duplicates** to ensure that there is no redundancy in the dataset, which could bias the model.

# Check for duplicate rows based on all columns

duplicates = df.duplicated()

print("\nNumber of duplicate rows:", duplicates.sum())

# Display duplicate rows (if any)

if duplicates.sum() > 0:

print("\nDuplicate Rows:")

print(df[duplicates])

# Step 3: Handle Duplicates

# Drop duplicate rows (keeping the first occurrence)

df = df.drop\_duplicates()

# Verify that duplicates have been removed

print("\nNumber of duplicate rows after handling:", df.duplicated().sum())

**Step 5: Visualize the Time Series Data**

* First visualization shows the daily sales trends over time.
* It helps identify patterns, fluctuations, or anomalies in sales on a day-to-day basis.
* Second visualization aggregates the daily sales data into monthly sales.
* It provides a smoother view of the data, making it easier to observe long-term trends and seasonality.

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

plt.plot(df\_sales.index, df\_sales['Sales'], label='Sales', color='blue')

plt.title('Sales Over Time')

plt.xlabel('Order Date')

plt.ylabel('Sales')

plt.legend()

plt.grid(True)

plt.show()

# Resample data to monthly frequency (if data is daily)

df\_resampled = df\_sales.resample('M').sum()

# Plot the resampled data

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

plt.plot(df\_resampled.index, df\_resampled['Sales'], label='Monthly Sales', color='green')

plt.title('Monthly Sales Over Time')

plt.xlabel('Order Date')

plt.ylabel('Sales')

plt.legend()

plt.grid(True)

plt.show()

**Step 6: Split Dataset into Training and Testing Sets**

* **Split** the dataset into training and testing sets, ensuring that the data is randomly divided but the distribution of the target variable (Sales) is maintained in both sets (stratified).
* This split ensures that you can evaluate the performance of your model effectively.

# Define the split ratio (e.g., 80% train, 20% test)

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

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

# Step 8: Save the Train and Test Sets (Optional)

train.to\_csv('train\_sales.csv')

test.to\_csv('test\_sales.csv')

print("\nTraining and Testing datasets saved as 'train\_sales.csv' and 'test\_sales.csv'.")

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A graph showing sales over time

AI-generated content may be incorrect.**

**A graph showing a green line

AI-generated content may be incorrect.**

**Result:**

Thus the programs for time series data cleaning, loading and handling times series data and pre-processing techniques has been implemented successfully.