Exp no: 1 Time series Data Cleaning , Loading and Handle Data preprocessing 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 Passengers 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 Passengers 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

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

```
# Step 1: Load the Dataset

file_path = '/content/sales.csv'  # Update with your actual file path

data = pd.read_csv(file_path)

# Check for null values

print("\nChecking for Missing Values:")

print(data.isnull().sum())
```

Step 2: Visualize Missing Values

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

```
# Step 2: Visualize Missing Values

import matplotlib.pyplot as plt

import missingno as msno

msno.matrix(data)

plt.title("Missing Values Matrix")

plt.show()
```

Step 3: Handle Missing or Invalid Values

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

```
# Step 3: Handle Missing or Invalid Values

# Replace invalid/missing values in numeric columns with NaN and fill with median

numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns

for col in numeric_columns:

    data[col] = data[col].replace(0, None)  # Replace 0s if considered invalid

data[numeric_columns] = data[numeric_columns].fillna(data[numeric_columns].median())
```

Step 4: Remove Duplicate Rows

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

```
# Step 4: Remove Duplicate Rows

duplicates = data.duplicated().sum()

print(f"\nNumber of Duplicate Rows: {duplicates}")

data = data.drop_duplicates()
```

```
print("\nCleaned Data Information:")

print(data.info())

print("\nSample Cleaned Data:\n", data.head())
```

Step 5: Feature Normalization and Target Separation

1. **Separate** the dataset into features (X) and target (y). In this case, the target column is assumed to be 'Sales'.
2. **Normalize** the numeric features to bring all values to a comparable scale (optional, as commented out in your code).

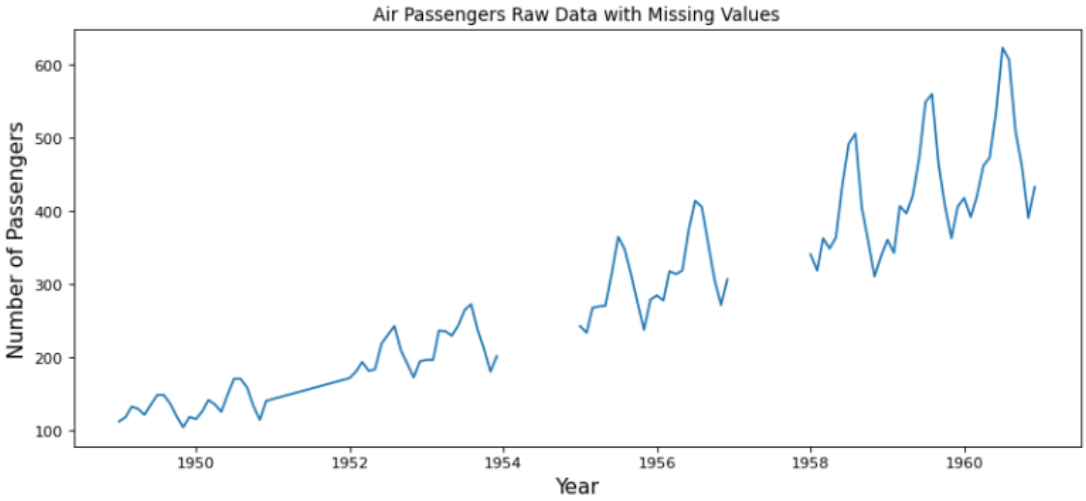
```
# Step 5: Feature Normalization and Target Separation

from sklearn.preprocessing import StandardScaler

target_column = 'Sales'

features = data.drop(columns=[target_column])  # Drop target column

target = data[target_column]

# Optional: Normalize numerical features

scaler = StandardScaler()

features[numeric_columns] = scaler.fit_transform(features[numeric_columns])
```

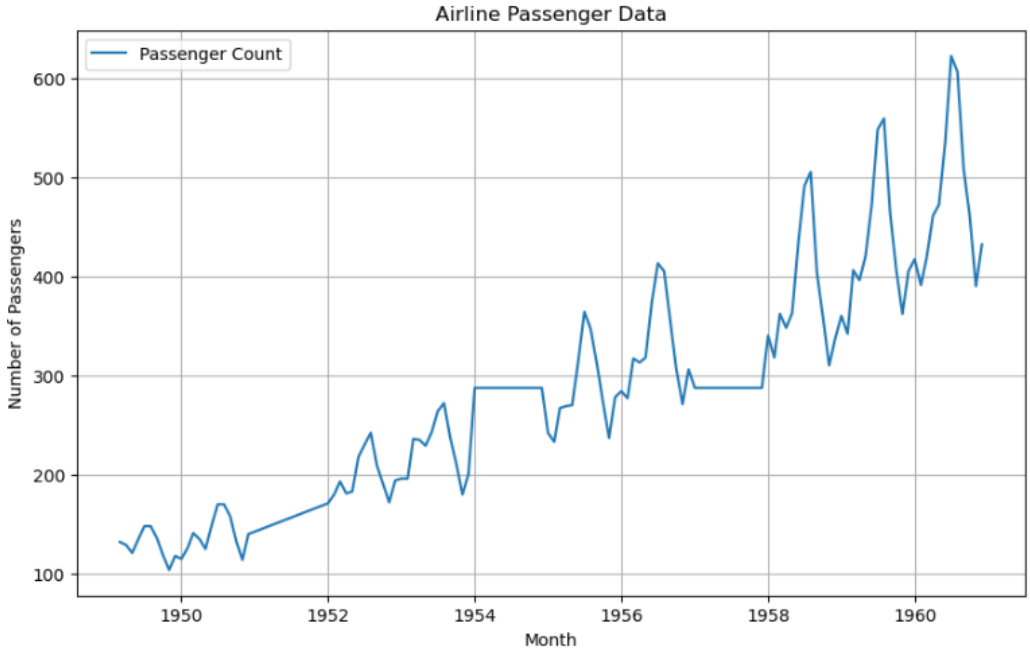
Step 6: Split Dataset into Training and Testing Sets

1. **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).
2. This split ensures that you can evaluate the performance of your model effectively.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Check the split

print(f"\nTraining Set Shape: X_train: {X_train.shape}, y_train: {y_train.shape}")

print(f"Testing Set Shape: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

Output :





Result :

The Time Series Program for data cleaning , pre-processing is Successfully Executed