import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

1. **Importing pandas**: pandas library is imported as pd, which is commonly used for data manipulation and analysis in Python.
2. **Importing Scikit-Learn Tools**: Various preprocessing tools from the sklearn library are imported:
   * SimpleImputer is used to handle missing values.
   * LabelEncoder is for converting categorical labels into numeric format.
   * OneHotEncoder is used for one-hot encoding categorical variables.
   * StandardScaler is to standardize features by removing the mean and scaling to unit variance.

Creating a Sample DataFrame

data = {

'A': [1, 2, None, 4, 5],

'B': ['X', None, 'Y', 'Z', 'X'],

'C': [7, 8, 9, None, 11]

}

df = pd.DataFrame(data)

print("DataSet:\n",df)

1. **Sample Data Creation**: A dictionary data is defined with keys A, B, and C. Here, A and C are numeric columns and B is a categorical column with some missing values (None).
2. **DataFrame Creation**: A pandas DataFrame df is created using the dictionary.
3. **Printing DataFrame**: The initial DataFrame is printed. It shows how the data looks before any manipulation.

Handling Missing Values

imputer = SimpleImputer(strategy='mean')

df[['A', 'C']] = imputer.fit\_transform(df[['A', 'C']])

print("DataSet after handling Missing Values of A and C Columns:\n",df[['A','C']])

1. **Creating an Imputer**: An instance of SimpleImputer is created with the strategy to replace missing values with the mean of the column.
2. **Filling Missing Values**: The fit\_transform method is used to compute the mean for columns A and C, and replace missing values with these means. The result is assigned back to the corresponding columns in df.
3. **Printing Updated DataFrame**: The updated DataFrame showing only columns A and C is printed to verify that missing values were replaced.

df.loc[df['B'].isna(), 'B'] = 'Unknown' *# Use loc to avoid chained assignment warning*

1. **Handling Missing Values in Column B**: The program uses loc to locate rows in column B where values are NaN (missing) and replaces them with the string 'Unknown'. Using loc helps to avoid warnings related to chained assignments in pandas.

Encoding Categorical Variables

label\_encoder = LabelEncoder()

df['B\_encoded'] = label\_encoder.fit\_transform(df['B'])

print("\nDataSet after handling Missing Values of B After Label encoding:\n", df['B\_encoded'])

1. **Creating a Label Encoder**: An instance of LabelEncoder is created to convert categorical labels into integers.
2. **Encoding Column B**: The fit\_transform method encodes the values in B, assigning each unique category an integer. The result is stored in a new column B\_encoded.
3. **Printing Encoded Column**: The encoded values of column B (now stored in B\_encoded) are printed.

encoded\_data = OneHotEncoder().fit\_transform(df[['B\_encoded']]).toarray()

encoded\_df = pd.DataFrame(encoded\_data, columns=[f'B\_{i}' for i in range(encoded\_data.shape[1])])

print("\nDataSet after handling Missing Values of B Before Label encoding:\n", df['B'])

1. **OneHot Encoding**: OneHotEncoder is used to convert the integer-encoded label in B\_encoded into a one-hot encoded format (binary columns). The result is converted to an array.
2. **Creating One-Hot Encoded DataFrame**: A new DataFrame encoded\_df is created with columns named according to the number of unique categories in B\_encoded.
3. **Printing Original Categorical Column Values**: The original values of column B are printed again for reference.

Combining DataFrames

Copydf = pd.concat([df, encoded\_df], axis=1)

print("DataSet after handling Missing Values of B After one\_hot\_encoder:\n",df)

1. **Combining DataFrames**: The one-hot encoded DataFrame encoded\_df is concatenated with the original DataFrame df along the columns (axis=1), effectively adding the new one-hot encoded columns to df.
2. **Printing Updated DataFrame**: The updated DataFrame is printed to show the effect of one-hot encoding.

Feature Scaling

scaled\_data = StandardScaler().fit\_transform(df[['A', 'C']])

scaled\_df = pd.DataFrame(scaled\_data, columns=['A\_scaled', 'C\_scaled'])

df = pd.concat([df, scaled\_df], axis=1)

1. **Feature Scaling**: An instance of StandardScaler is created to scale numeric features A and C. The fit\_transform method standardizes these features (removing the mean and scaling to unit variance).
2. **Creating Scaled DataFrame**: A new DataFrame scaled\_df is created from the scaled values, with clear column names A\_scaled and C\_scaled.
3. **Combining with Original DataFrame**: The new scaled DataFrame is concatenated with the original DataFrame df, adding the scaled features to it.

Display Final DataFrame

print("Feature Scaling using Standard scaler\n", df)

1. **Final Print Statement**: The final state of the DataFrame df is printed to show the complete dataset after imputation, encoding, and scaling.