import numpy as np

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

"""""

desired\_width = 320

pd.set\_option('display.width',desired\_width)

pd.set\_option('display.max\_columns',14)

"""

heart = pd.read\_csv('E:\SEMESTER\Summer- 21\CSE422\Lab\lab-05\heart failur classification dataset.csv')

#heart.insull()

#printing 1st 7 rows

print(heart.head(7))

#printing total rows & column

print(heart.shape)

# To get the column info with missing data

print(heart.isnull().sum())

#Handling missing values

#For serum\_sodium column

print()

print("Handling missing values:")

# Check how many values are missing in the serum\_sodium column

print("Number of rows with null values in serum\_sodium column: ", heart['serum\_sodium'].isnull().sum())

heart = heart[heart['serum\_sodium'].notnull()]

# Print out the shape of the heart

print("Shape after removing null values in serum\_sodium: ", heart.shape)

#For time column

# Check how many values are missing in the time column

print("Number of rows with null values in time column: ", heart['time'].isnull().sum())

# Subset the volunteer dataset

heart = heart[heart['time'].notnull()]

print("Shape after removing null values in time: ", heart.shape)

"""""

#Imputing missing values

#from sklearn.impute import SimpleImputer

impute = SimpleImputer(missing\_values=np.nan, strategy='mean')

impute.fit(heart[['serum\_sodium']])

heart['serum\_sodium'] = impute.transform(heart[['serum\_sodium']])

#print(heart.isnull().sum())

print("After imputing values in serum\_sodium column the shape is:",heart.shape)"""

#Encoding categorical features

print()

print("Encoding categorical features part:")

print(heart.info())

print(heart['sex'].unique())

# Set up the LabelEncoder object

enc = LabelEncoder()

# Apply the encoding to the "Accessible" column

heart['sex\_enc'] = enc.fit\_transform(heart['sex'])

# Compare the two columns

print(heart[['sex', 'sex\_enc']].head(7))

print(heart['smoking'].unique())

enc = LabelEncoder()

# Apply the encoding to the "Accessible" column

heart['smoking\_enc'] = enc.fit\_transform(heart['smoking'])

# Compare the two columns

print(heart[['smoking', 'smoking\_enc']].head(7))

print(heart.head(33))

print(heart.isnull().sum())

#dropping column

heart = heart.drop(['sex'], axis = 1)

heart = heart.drop(['smoking'], axis = 1)

print("After dropping time column the shape is: ",heart.shape)

print()

print("Scaling all the values between 0-1 with proper scaling technique:")

#from sklearn.datasets import load\_breast\_cancer

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

heart\_scaling = heart

X\_train, X\_test, y\_train, y\_test = train\_test\_split(heart\_scaling, heart\_scaling.values, random\_state=1)

print(X\_train.shape)

print(X\_test.shape)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(X\_train)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(X\_train)

# transform data

X\_train\_scaled = scaler.transform(X\_train)

print("per-feature minimum before scaling:\n {}".format(X\_train.min(axis=0)))

print("per-feature maximum before scaling:\n {}".format(X\_train.max(axis=0)))

print("per-feature minimum after scaling:\n {}".format(X\_train\_scaled.min(axis=0)))

print("per-feature maximum after scaling:\n {}".format(X\_train\_scaled.max(axis=0)))

# transform test data

X\_test\_scaled = scaler.transform(X\_test)

print(X\_test\_scaled)

#Feature Engineering:

print()

print("Feature Engineering:")

label = heart.DEATH\_EVENT

print("label:\n{}".format(label.head(19)))

to\_drop = ["DEATH\_EVENT"]

features = heart.drop(to\_drop, axis=1)

#print(features.head(7))

scaler = MinMaxScaler()

scaler.fit(features)

# transform data

feature\_train\_scaled = scaler.transform(features)

feature\_train\_df = pd.DataFrame(feature\_train\_scaled)

print(feature\_train\_df.head(5))