Exp No: 1	Setting Up the Environment And Preprocessing the Data

Aim:

To set up a fully functional machine learning development environment and to perform data preprocessing operations like handling missing values, encoding categorical variables, feature scaling, and splitting datasets.

Algorithm:

1. Install Required Libraries:
    - Install numpy, pandas, matplotlib, seaborn, and scikit-learn using pip.
2. Import Libraries.
3. Load Dataset:
    - Load any dataset (e.g., Titanic or Iris) using pandas.
4. Data Exploration:
    - Use df.info(), df.describe(), df.isnull().sum() to understand the data.
5. Handle Missing Values:
    - Use .fillna() or .dropna() depending on the strategy.
6. Encode Categorical Data:
    - Use pd.get_dummies() or LabelEncoder.
7. Feature Scaling:
    - Normalize or standardize the numerical features using StandardScaler or MinMaxScaler.
8. Split Dataset:
    - Use train_test_split() from sklearn to create training and testing sets.
9. Display the Preprocessed Data.

Code:

```
# 1. Install necessary libraries (if not already installed)
# !pip install numpy pandas matplotlib seaborn scikit-learn

# 2. Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

# 3. Load dataset
df = sns.load_dataset('titanic')  # Titanic dataset
df.head()

# 4. Explore the dataset
print(df.info())
print(df.describe())
print(df.isnull().sum())

# 5. Handle missing values
# Fill age with median, embark_town with mode
df['age'].fillna(df['age'].median(), inplace=True)
df['embark_town'].fillna(df['embark_town'].mode()[0], inplace=True)
df.drop(columns=['deck'], inplace=True)  # too many missing values

# 6. Encode categorical variables
# Convert 'sex' and 'embark_town' using LabelEncoder
```

```
le = LabelEncoder()
df[‘sex’] = le.fit_transform(df[‘sex’])
df[‘embark_town’] = le.fit_transform(df[‘embark_town’])

# Drop non-informative or redundant columns
df.drop(columns=[‘embarked’, ‘class’, ‘who’, ‘alive’, ‘adult_male’, ‘alone’], inplace=True)

# 7. Feature Scaling
scaler = StandardScaler()
numerical_cols = [‘age’, ‘fare’]
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# 8. Split dataset
# Define features (X) and label (y)
X = df.drop(‘survived’, axis=1)
y = df[‘survived’]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 9. Show final preprocessed data
print(“Training Data Shape:”, X_train.shape)
print(“Test Data Shape:”, X_test.shape)
X_train.head()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
```

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```
Training Data Shape: (712, 7)
Test Data Shape: (179, 7)
/tmp/ipython-input-4068659829.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['age'].fillna(df['age'].median(), inplace=True)
/tmp/ipython-input-4068659829.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.
```

```
df['embark_town'].fillna(df['embark_town'].mode()[0], inplace=True)
```

	pclass	sex	age	sibsp	parch	fare	embark_town
331	1	1	1.240235	0	0	-0.074583	2
733	2	1	-0.488887	0	0	-0.386671	2
382	3	1	0.202762	0	0	-0.488854	2
704	3	1	-0.258337	1	0	-0.490280	2
813	3	0	-1.795334	4	2	-0.018709	2