# Experiment :

**AIM :** Performing data preprocessing on different datasets

## THEORY :

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

Why we use Data Preprocessing ?

In Real world data are generally incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data. Noisy: containing errors or outliers. Inconsistent: containing discrepancies in codes or names.

**Steps in Data Preprocessing**

**Step 1 :** Import the libraries
**Step 2 :** Import the data-set
**Step 3 :** Check out the missing values
**Step 4 :** See the Categorical Values
**Step 5 :** Splitting the data-set into Training and Test Set
**Step 6 :** Feature Scaling

**Step 1 : Import the Libraries**

Import the Libraries

This is how we import libraries in Python using import keyword and this is the most popular libraries which any Data Scientist used.

- **NumPy** is the fundamental package for scientific computing with Python. It contains among other things:

1. A powerful N-dimensional array object
2. Sophisticated (broadcasting) functions
3. Tools for integrating C/C++ and FORTRAN code
4. Useful linear algebra, Fourier transform, and random number capabilities

- **Pandas** is for data manipulation and analysis. Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

- **Matplotlib** is a Python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms.

- **Seaborn** is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Step 2 : Import the Data-set**

Import the Dataset

By using Pandas we import our data-set .[Note: It's not necessarily every-time you deal with **CSV** file, sometimes you deal with **Html or Xlsx(Excel file)** ]. However, to access and to use fastly we use CSV files because of their light weights. After importing the dataset, you can see we use head function ( This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it. By default it returns 5 rows. )

**Step 3 : Check out the Missing Values**

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly by the researcher, then he/she may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained by the researcher will differ from ones where the missing values are present.

Missing Values

**Two ways to handle Missing Values**

1.DELETE THE MISSING VALUES : This method commonly used to handle the null values. Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 75% of missing values. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss

2.REPLACE THE MISSING VALUES : This strategy can be applied on a feature which has numeric data like the year column or Home team goal column. We can calculate the **mean, median or mode** of the feature and replace it with the missing values. This is an approximation which can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns. Replacing with the above three approximations are a statistical approach of handling the missing values. This method is also called as **leaking the data** while training. Another way is to approximate it with the deviation of neighbouring values. This works better if the data is linear.

Above strategy is good for numeric data. But what happen when Categorical data has missing values?

**Step 4 : Managing the Categorical Values**

Since, machine learning models are based on Mathematical equations and you can intuitively understand that it would cause some problem if we can keep the Categorical data in the equations because we would only want numbers in the equations.
So, we need to encode the Categorical Variable. To convert Categorical variable into Numerical data we can use LabelEncoder() class from preprocessing library.
Use LabelEncoder class to convert Categorical data into numerical one
label_encoder is object which is I use and help us in transferring Categorical data into Numerical data. Next, I fitted this label_encoder object to the first column of our matrix X and all this return the first column country of the matrix X encoded.
But there is a problem in it, the problem is still the same, machine learning models are based on equations and that's good that we replaced the text by numbers so that we can include the numbers in the equations.
However, since 1>0 and 2>1(See the above data-set) , the equations in the model will think that Spain has a higher value than Germany and France, and Germany has a higher value than France. Actually, this is a not the case, these are actually three Categories and there is no relational order between the three. So , we have to prevent this, we're going to use what are **Dummy Variables.**
**What is Dummy Variables ?**

**Dummy Variables** is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome.To avoid dummy variable trap we do **Number of Columns = Number of Categories-1**

To create dummy variable we can use OneHotEncoder Class from sklearn.preprocessing or you can use pandas get dummies method.

Sometimes, we use **KNN Imputation(for Categorical variables):** In this method of imputation, the missing values of an attribute are imputed using the given number of attributes that are most similar to the attribute whose values are missing. The similarity of two attributes is determined using a distance function.

**Step 5 : Splitting the data-set into Training and Test Set**
In any Machine Learning model is that we're going to split data-set into two separate sets
**1. Training Set**

**2. Test Set**

**Why we need splitting ?**

Well here it's your algorithm model that is going to learn from your data to make predictions. Generally we split the data-set into 70:30 ratio or 80:20 what does it mean, 70 percent data take in train and 30 percent data take in test. However, this Splitting can be varies according to the data-set shape and size.

Splitting the Data-set into two set—Train and Test Set

X_train is the training part of the matrix of features.

X_test is the test part of the matrix of features.

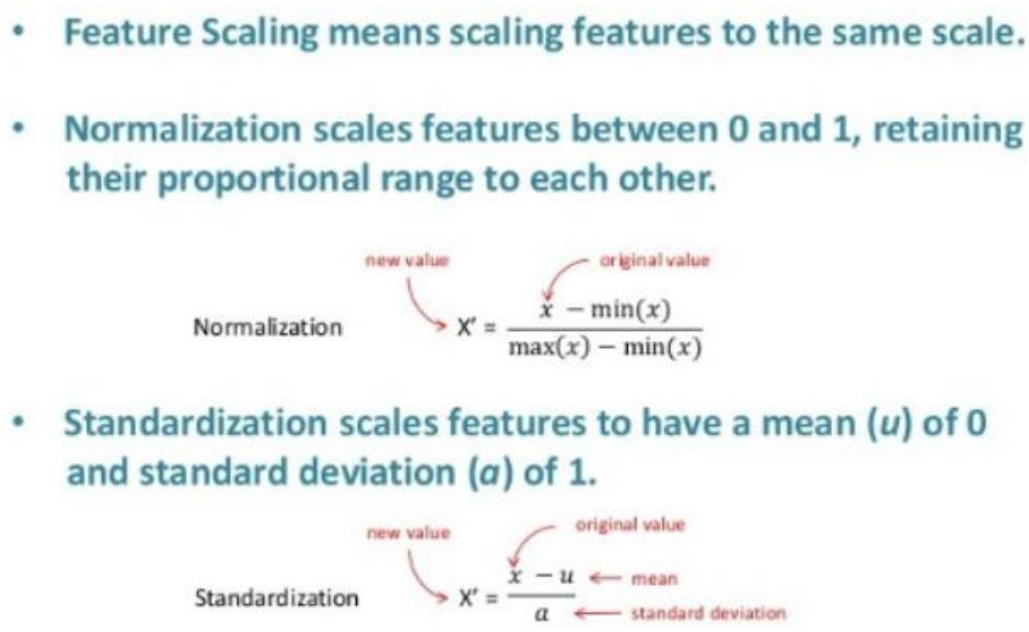
y_train is the training part of the dependent variable that is associated to X_train here.

y_test is the test part of the dependent variable that is associated to X_train here.

**Step 6 : Feature Scaling**

**What is Feature Scaling ?**

**Feature scaling** is the method to limit the range of variables so that they can be compared on common grounds.



## CODE :

In [1]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Importing the dataset
dataset = pd.read_csv('dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
```

In [3]: dataset

Out[3]:

| | School | Age | PocketMoney | Smokes |
|---|---|---|---|---|
| 0 | Middle | 14.0 | 50.0 | No |
| 1 | Secondary | 19.0 | 49.0 | Yes |
| 2 | High | 16.0 | 98.0 | No |
| 3 | Secondary | 18.0 | 62.0 | No |
| 4 | High | 17.0 | NaN | Yes |
| 5 | Middle | 15.0 | 89.0 | Yes |
| 6 | Secondary | NaN | 20.0 | No |
| 7 | Middle | 14.0 | 46.0 | Yes |
| 8 | High | 16.0 | 82.0 | No |
| 9 | Middle | 13.0 | 34.0 | Yes |

In [4]: X

Out[4]:
```
array([['Middle', 14.0, 50.0],
       ['Secondary', 19.0, 49.0],
       ['High', 16.0, 98.0],
       ['Secondary', 18.0, 62.0],
       ['High', 17.0, nan],
       ['Middle', 15.0, 89.0],
       ['Secondary', nan, 20.0],
       ['Middle', 14.0, 46.0],
       ['High', 16.0, 82.0],
       ['Middle', 13.0, 34.0]], dtype=object)
```

In [5]: y

Out[5]: array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'], dtype=object)

In [6]: # Taking care of missing data
```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

In [7]: X

Out[7]: array([['Middle', 14.0, 50.0],
['Secondary', 19.0, 49.0],
['High', 16.0, 98.0],
['Secondary', 18.0, 62.0],
['High', 17.0, 58.888888888888886],
['Middle', 15.0, 89.0],
['Secondary', 15.777777777777779, 20.0],
['Middle', 14.0, 46.0],
['High', 16.0, 82.0],
['Middle', 13.0, 34.0]], dtype=object)

In [8]: # Encoding categorical data
```
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])

# Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

In [9]: X

Out[9]: array([[1, 14.0, 50.0],
[2, 19.0, 49.0],
[0, 16.0, 98.0],
[2, 18.0, 62.0],
[0, 17.0, 58.888888888888886],
[1, 15.0, 89.0],
[2, 15.777777777777779, 20.0],
[1, 14.0, 46.0],
[0, 16.0, 82.0],
[1, 13.0, 34.0]], dtype=object)

In [10]: y

Out[10]: array([0,1, 0, 0, 1, 1, 0, 1, 0, 1], dtype=int64)

```
In [11]: onehotencoder = OneHotEncoder(categorical_features = [0])
         X = onehotencoder.fit_transform(X).toarray()

In [12]: X

Out[12]: array([[ 0.        ,  1.        ,  0.        , 14.        , 50.        ],
                [ 0.        ,  0.        ,  1.        , 19.        , 49.        ],
                [ 1.        ,  0.        ,  0.        , 16.        , 98.        ],
                [ 0.        ,  0.        ,  1.        , 18.        , 62.        ],
                [ 1.        ,  0.        ,  0.        , 17.        , 58.88888889],
                [ 0.        ,  1.        ,  0.        , 15.        , 89.        ],
                [ 0.        ,  0.        ,  1.        , 15.77777778, 20.        ],
                [ 0.        ,  1.        ,  0.        , 14.        , 46.        ],
                [ 1.        ,  0.        ,  0.        , 16.        , 82.        ],
                [ 0.        ,  1.        ,  0.        , 13.        , 34.        ]])

In [13]: #Avoiding dummy variable trap
         X=X[:,1:5]

In [14]: X

Out[14]: array([[ 1.        ,  0.        , 14.        , 50.        ],
                [ 0.        ,  1.        , 19.        , 49.        ],
                [ 0.        ,  0.        , 16.        , 98.        ],
                [ 0.        ,  1.        , 18.        , 62.        ],
                [ 0.        ,  0.        , 17.        , 58.88888889],
                [ 1.        ,  0.        , 15.        , 89.        ],
                [ 0.        ,  1.        , 15.77777778, 20.        ],
                [ 1.        ,  0.        , 14.        , 46.        ],
                [ 0.        ,  0.        , 16.        , 82.        ],
                [ 1.        ,  0.        , 13.        , 34.        ]])

In [15]: # Splitting the dataset into the Training set and Test set
         from sklearn.cross_validation import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_sta


In [16]: X_train

Out[16]: array([[ 0.        ,  0.        , 17.        , 58.88888889],
                [ 1.        ,  0.        , 13.        , 34.        ],
                [ 0.        ,  1.        , 19.        , 49.        ],
                [ 0.        ,  1.        , 15.77777778, 20.        ],
                [ 1.        ,  0.        , 14.        , 46.        ],
                [ 0.        ,  1.        , 18.        , 62.        ],
                [ 1.        ,  0.        , 14.        , 50.        ],
                [ 1.        ,  0.        , 15.        , 89.        ]])
```

In [17]: y_train

Out[17]: array([1, 1, 1, 0, 1, 0, 0, 1], dtype=int64)

In [18]: X_test

Out[18]: array([[ 0.,0., 16., 98.],
[ 0.,0., 16., 82.]])

In [19]: y_test

Out[19]: array([0, 0], dtype=int64)

In [20]: # Feature Scaling
**from sklearn.preprocessing import** StandardScaler
sc_X = StandardScaler()
X_train[:,2:] = sc_X.fit_transform(X_train[:,2:])
X_test[:,2:] = sc_X.transform(X_test[:,2:])

In [21]: X_train

Out[21]: array([[ 0. , 0. , 0.64463463, 0.40800723],
[ 1. , 0. , -1.37335204, -0.89761592],
[ 0. , 1. , 1.65362797, -0.11074482],
[ 0. , 1. , 0.02802759, -1.63202894],
[ 1. , 0. , -0.86885537, -0.26811904],
[ 0. , 1. , 1.1491313 , 0.57121013],
[ 1. , 0. , -0.86885537, -0.05828675],
[ 1. , 0. , -0.36435871, 1.9875781 ]])

In [22]: X_test

Out[22]: array([[0. , 0. , 0.14013796, 2.45970076],
[0. , 0. , 0.14013796, 1.62037159]])

## DISCUSSION :

In Real world data are generally

- Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data.
- Noisy: containing errors or outliers.
- Inconsistent: containing discrepancies in codes or names.

We use data preprocessing to convert our data into usable form by removing above errors, and we can do it very easily as above in using python programming language.

## FINDING AND LEARNING :

Data pre-processing refers to the transformations applied to your data before feeding it to the algorithm. In python, scikit-learn library has a pre-built functionality under sklearn.preprocessing. After applying data preprocessing our machine learning algorithm is able to work well on our dataset, Data preprocessing improves accuracy, speed and understandability of our machine learning model.