Data Pre-Processing:

Data pre-processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis.

Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data pre-processing Follow the following steps to process your Data

- Import the Libraries
- Importing the dataset
- Taking care of Missing Data
- Label encoding
- One Hot Encoding
- Feature Scaling
- Splitting Data into Train and Test

Step1: Importing the libraries

First step is usually importing the libraries that will be needed in the program. Import the pandas library and give a shortcut name as pd

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Step 2: Import the Dataset

We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program) and read it using a method called read_csv which can be found in the library called pandas. Here we are using a data set which you can find in the below link:

https://thesmartbridge.com/documents/spsaimldocs/Data.csv

```
dataset = pd.read_csv(
r'C:\Users\Hari Chandan\Desktop\Data_Preprocessing\Data.csv')
```

Step 3: Taking Care of missing Data

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values of the same column and have it to replace the missing data.

We will be suing dataset.isnull().any() method to see which column has missing values.

```
dataset.isnull().any()
```

```
Country      False
Age           True
Salary        True
Purchased    False
dtype: bool
```

Word “True” that the particular column has missing values ie, in our dataset age and salary column has missing values

We can replace the missing values by, mean median or mode by using fillna method. The arguments that are to be passed are as follows:

```
dataset.fillna(dataset.mean(),inplace=True)
```

If you apply this to dataset where ever you find the missing values in any column that will be replaced.

Or you can also specify an individual column ,which is as follows

```
dataset['Age'].fillna(dataset['Age'].mean(),inplace=True)
dataset['Salary'].fillna(dataset['Salary'].mean(),inplace=True)
```

If you find textual data in the respective colum then use mode to replace the missing values

Step 4: Label Encoding

Sometimes in the dataset we will find textual data like names, countries states, then the machine cannot do mathematical operations or cannot understand the textual data. So the textual data are to be converted in to numerical format which is called as label encoding. we make use of label Encoder class to convert textual data in to Numerical data. In the given dataset country has textual data so we will be converting that particular columns textual data to numerical values.

```
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
dataset['Country'] = labelencoder_y.fit_transform(dataset['Country'])
```

You have to apply this for every column which has textual data

Now the country column with country names will be converted in to numerical values. These numerical values are assigned based on alphabetical order. In the data set we have three countries France Spain and Germany. France is assigned with 0, Germany is assigned with 1 and Spain is assigned with 2.

If you consider mathematical expressions, then number 2 will be greater than 1 and 1 will be greater than 0. But here countries are assigned with numbers whenever your machine get this number then it will think Germany is greater than japan or vice versa so we have to encode these numerical values in to binary format so that our machine will not consider any priority to the country.

Before converting into binary format lets split the data set in to independent and dependent variable

Step 4: Splitting Dataset in to Independent variable and Dependent variable

To read the columns, we will use iloc of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

```
x = dataset.iloc[:, 0:3].values
y = dataset.iloc[:, 3].values
```

From the above piece of code **":"** indicates you are considering all the rows and **"0:3"** indicates we are considering column 0 to 2 as input values and assigning them to variable x. in the same way in second line **":"** indicates you are considering all the rows and **"3"** indicates we are considering only one column 3 as output value and assigning them to variable y

Step 5: One Hot Encoding

As discussed in step3 we are converting numerical data in to binary data. This is what happens when you binaries your data

Country	X[0]	X[1]	X[2]
Spain	0	0	1
Germany	1	0	0
France	0	1	0

In the above table extra three columns are created. Based on the categories those many columns will be appended to the x variable

To accomplish the task, we will import yet another library called OneHotEncoder.

Next we will create an object of that class, as usual, and assign it to onehotencoder. OneHotEncoder takes an important parameter called categorical_features which takes the value of the index of the column of categories.

```
from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder(categorical_features = [0])
```

The code above will select the first column to OneHotEncode the categories. we will use fit_transform OneHotEncoder and additionally include toarray()

```
X = onehotencoder.fit_transform(X).toarray()
```

If you check your dataset now, all your categories will have been encoded to 0s and 1s.

we should remove the dummy variable with following syntax

```
X = x[:,1:]
```

The above piece of code will vomit the first column keeping all the other columns

Step 6: Splitting The dataset in to Train set and Testing set

Now we need to split our dataset into two sets — a Training set and a Test set. A general rule of the thumb is to allocate 80% of the dataset to training set and the remaining 20% to test set. For this task, we will import test_train_split from model_selection library of scikit.

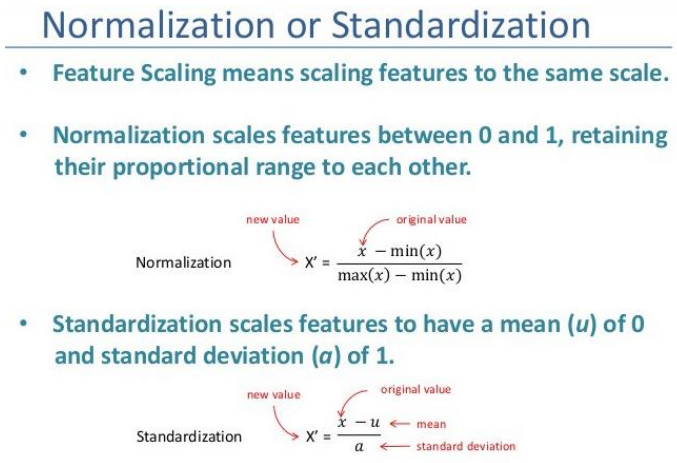
Now to build our training and test sets, we will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices) , Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices). We will assign to them the test_train_split, which takes the parameters — arrays (X and Y), test_size (if we give it the value 0.5, meaning 50%, it would split the dataset into half. Since an ideal choice is to allocate 20% of the dataset to test set, it is usually assigned as 0.2. 0.25 would mean 25%, just saying).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size = 0.2, random_state = 0)
```

Step 7: Feature scaling:

The final step of data preprocessing is to apply the very important feature scaling. It is a method used to standardize the range of independent variables or features of data.

A lot of machine learning models are based on Euclidean distance. If, for example, the values in one column (x) is much higher than the value in another column (y), (x2-x1) squared will give a far greater value than (y2-y1) squared. So clearly, one square difference dominates over the other square difference. In the machine learning equations, the square difference with the lower value in comparison to the far greater value will almost be treated as if it does not exist. We do not want that to happen. That is why it is necessary to transform all our variables into the same scale. There are several ways of scaling the data. One way is called Standardization which may be used. For every observation of the selected column, our program will apply the formula of standardization and fit it to a scale.



To accomplish the job, we will import the class StandardScaler from the sckit preprocessing library and as usual create an object of that class.

Now we will fit and transform our X_train set (It is important to note that when applying the Standard Scalar object on our training and test sets, we can simply transform our test set but for our training set we have to at first fit it and then transform the set). That will transform all the data to a same standardized scale.

```
from sklearn.preprocessing import StandardScaler
sc1 = StandardScaler()
x_train = sc1.fit_transform(x_train)
x_test = sc1.transform(x_test)
```

These are the general 7 steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.