**Data Preprocessing in Machine Learning**

**Data preprocessing in Machine Learning is a crucial step that helps enhance the quality of data to promote the extraction of meaningful insights from the data. Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a data mining technique that transforms raw data into an understandable and readable format.**

***Why Data Preprocessing in Machine Learning?***

**When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models. Let’s explore various steps of data preprocessing in machine learning.**

***Steps in Data Preprocessing in Machine Learning***

**1. Acquire the dataset**

**Acquiring the dataset is the first step in data preprocessing in machine learning. To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according to use cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medical dataset will include healthcare-related data.**

**There are several online sources from where you can download datasets like**[**https://www.kaggle.com/uciml/datasets**](https://www.kaggle.com/uciml/datasets)**and**[**https://archive.ics.uci.edu/ml/index.php**](https://archive.ics.uci.edu/ml/index.php)**.**

**You can also create a dataset by collecting data via different Python APIs. Once the dataset is ready, you must put it in CSV, or HTML, or XLSX file formats.**

**2. Import all the crucial libraries**

**Since Python is the most extensively used and also the most preferred library by Data Scientists around the world.**

**The predefined Python libraries can perform specific data preprocessing jobs. Importing all the crucial libraries is the second step in data preprocessing in machine learning.**

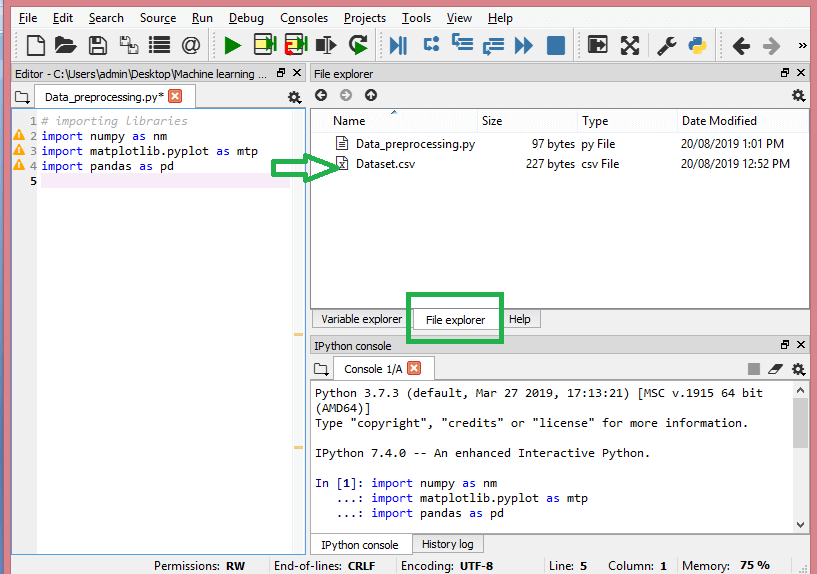
**The three core Python libraries used for this data preprocessing in Machine Learning are:**

* **NumPy – NumPy is the fundamental package for scientific calculation in Python. Hence, it is used for inserting any type of mathematical operation in the code. Using NumPy, you can also add large multidimensional arrays and matrices in your code.**
* **Pandas – Pandas is an excellent open-source Python library for data manipulation and analysis. It is extensively used for importing and managing the datasets. It packs in high-performance, easy-to-use data structures and data analysis tools for Python.**
* **Matplotlib – Matplotlib is a Python 2D plotting library that is used to plot any type of charts in Python. It can deliver publication-quality figures in numerous hard copy formats and interactive environments across platforms (IPython shells, Jupyter notebook, web application servers, etc.).**

**3. Import the dataset**

**In this step, you need to import the dataset/s that you have gathered for the ML project at hand. Importing the dataset is one of the important steps in data preprocessing in machine learning. However, before you can import the dataset/s, you must set the current directory as the working directory. You can set the working directory in Spyder IDE in three simple steps:**

1. **Save your Python file in the directory containing the dataset.**
2. **Go to File Explorer option in Spyder IDE and choose the required directory.**
3. **Now, click on the F5 button or Run option to execute the file.**



**This is how the working directory should look.**

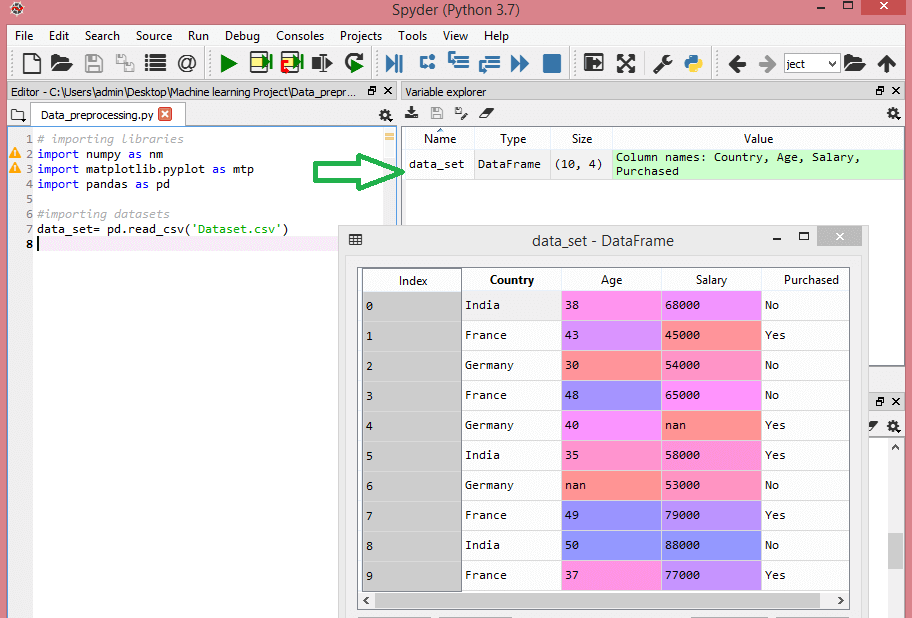
**Once you’ve set the working directory containing the relevant dataset, you can import the dataset using the “read\_csv()” function of the Pandas library. This function can read a CSV file (either locally or through a URL) and also perform various operations on it. The read\_csv() is written as:**

**data\_set= pd.read\_csv(‘Dataset.csv’)**

**In this line of code, “data\_set” denotes the name of the variable wherein you stored the dataset. The function contains the name of the dataset as well. Once you execute this code, the dataset will be successfully imported.**

**During the dataset importing process, there’s another essential thing you must do – extracting dependent and independent variables. For every Machine Learning model, it is necessary to separate the independent variables (matrix of features) and dependent variables in a dataset.**

**Consider this dataset:**



**This dataset contains three independent variables – country, age, and salary, and one dependent variable – purchased.**

**How to extract the independent variables?**

**To extract the independent variables, you can use “iloc[ ]” function of the Pandas library. This function can extract selected rows and columns from the dataset.**

**x= data\_set.iloc[:,:-1].values**

**In the line of code above, the first colon(:) considers all the rows and the second colon(:) considers all the columns. The code contains “:-1” since you have to leave out the last column containing the dependent variable.**

**Run the code in your editor and see the output.**

**How to extract the dependent variable?**

**You can use the “iloc[ ]” function to extract the dependent variable as well. Here’s how you write it:**

**y= data\_set.iloc[:,3].values**

**This line of code considers all the rows with the last column only.**

**Run the code and see the output**

**4. Identifying and handling the missing values**

**In data preprocessing, it is pivotal to identify and correctly handle the missing values, failing to do this, you might draw inaccurate and faulty conclusions and inferences from the data. Needless to say, this will hamper your ML project.**

**Basically, there are two ways to handle missing data:**

* **Deleting a particular row – In this method, you remove a specific row that has a null value for a feature or a particular column where more than 75% of the values are missing. However, this method is not 100% efficient, and it is recommended that you use it only when the dataset has adequate samples. You must ensure that after deleting the data, there remains no addition of bias.**
* **Calculating the mean – This method is useful for features having numeric data like age, salary, year, etc. Here, you can calculate the mean, median, or mode of a particular feature or column or row that contains a missing value and replace the result for the missing value. This method can add variance to the dataset, and any loss of data can be efficiently negated. Hence, it yields better results compared to the first method (omission of rows/columns). Another way of approximation is through the deviation of neighbouring values. However, this works best for linear data.**

**5. Encoding the categorical data**

**Categorical data refers to the information that has specific categories within the dataset. In the dataset cited above, there are two categorical variables – country and purchased.**

**Machine Learning models are primarily based on mathematical equations. Thus, you can intuitively understand that keeping the categorical data in the equation will cause certain issues since you would only need numbers in the equations.**

**How to encode the country variable?**

**As seen in our dataset example, the country column will cause problems, so you must convert it into numerical values. To do so, you can use the LabelEncoder() class from the sci-kit learn library. The code will be as follows –**

**#Catgorical data**

**#for Country Variable**

**from sklearn.preprocessing import LabelEncoder**

**label\_encoder\_x= LabelEncoder()**

**x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])**

**And the output will be –**

**Here we can see that the LabelEncoder class has successfully encoded the variables into digits. However, there are country variables that are encoded as 0, 1, and 2 in the output shown above. So, the ML model may assume that there is come some correlation between the three variables, thereby producing faulty output. To eliminate this issue, we will now use Dummy Encoding.**

**Dummy variables are those that take the values 0 or 1 to indicate the absence or presence of a specific categorical effect that can shift the outcome. In this case, the value 1 indicates the presence of that variable in a particular column while the other variables become of value 0. In dummy encoding, the number of columns equals the number of categories.**

**Since our dataset has three categories, it will produce three columns having the values 0 and 1. For Dummy Encoding, we will use OneHotEncoder class of the scikit-learn library. The input code will be as follows –**

**#for Country Variable**

**from sklearn.preprocessing import LabelEncoder, OneHotEncoder**

**label\_encoder\_x= LabelEncoder()**

**x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])**

**#Encoding for dummy variables**

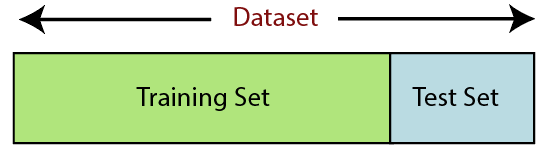
**onehot\_encoder= OneHotEncoder(categorical\_features= [0])**

**x= onehot\_encoder.fit\_transform(x).toarray()**

**Run the code and see the output.**

**6. Splitting the dataset**

**Splitting the dataset is the next step in data preprocessing in machine learning. Every dataset for Machine Learning model must be split into two separate sets – training set and test set.**



**Training set denotes the subset of a dataset that is used for training the machine learning model. Here, you are already aware of the output. A test set, on the other hand, is the subset of the dataset that is used for testing the machine learning model. The ML model uses the test set to predict outcomes.**

**Usually, the dataset is split into 70:30 ratio or 80:20 ratio. This means that you either take 70% or 80% of the data for training the model while leaving out the rest 30% or 20%. The splitting process varies according to the shape and size of the dataset in question.**

**To split the dataset, you have to write the following line of code –**

**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)**

**Here, the first line splits the arrays of the dataset into random train and test subsets. The second line of code includes four variables:**

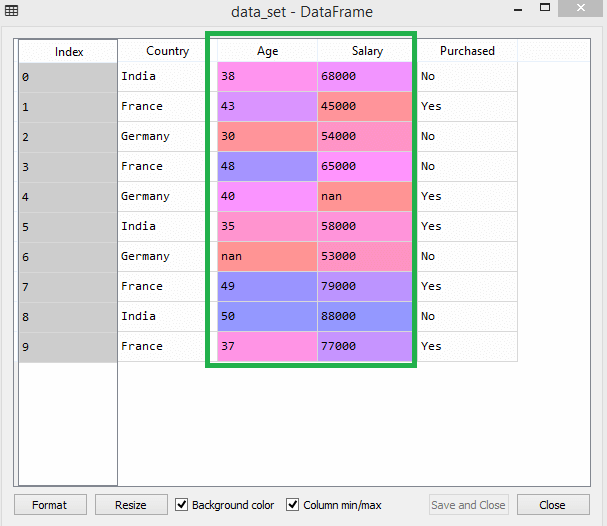
* **x\_train – features for the training data**
* **x\_test – features for the test data**
* **y\_train – dependent variables for training data**
* **y\_test – independent variable for testing data**

**Thus, the train\_test\_split() function includes four parameters, the first two of which are for arrays of data. The test\_size function specifies the size of the test set. The test\_size maybe .5, .3, or .2 – this specifies the dividing ratio between the training and test sets. The last parameter, “random\_state” sets seed for a random generator so that the output is always the same.**

**7. Feature scaling**

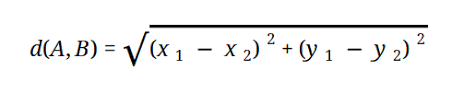
**Feature scaling marks the end of the data preprocessing in Machine Learning. It is a method to standardize the independent variables of a dataset within a specific range. In other words, feature scaling limits the range of variables so that you can compare them on common grounds.**

**Consider this dataset for example –**



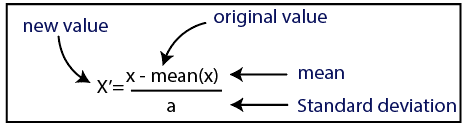
**In the dataset, you can notice that the age and salary columns do not have the same scale. In such a scenario, if you compute any two values from the age and salary columns, the salary values will dominate the age values and deliver incorrect results. Thus, you must remove this issue by performing feature scaling for Machine Learning.**

**Most ML models are based on Euclidean Distance, which is represented as:**

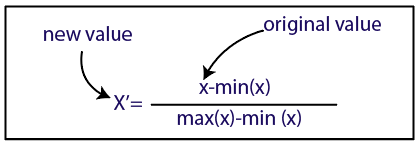


**You can perform feature scaling in Machine Learning in two ways:**

**Standardization**



**Normalization**



**For our dataset, we will use the standardization method. To do so, we will import StandardScaler class of the sci-kit-learn library using the following line of code:**

**from sklearn.preprocessing import StandardScaler**

**The next step will be to create the object of StandardScaler class for independent variables. After this, you can fit and transform the training dataset using the following code:**

**st\_x= StandardScaler()**

**x\_train= st\_x.fit\_transform(x\_train)**

**For the test dataset, you can directly apply transform() function (you need not use the fit\_transform() function because it is already done in training set). The code will be as follows –**

**x\_test= st\_x.transform(x\_test)**

**Run the code and check for x\_train and x\_test.**

**All the variables in the output are scaled between the values -1 and 1.**

**Now, to combine all the steps.**

**# importing libraries**

**import numpy as nm**

**import matplotlib.pyplot as mtp**

**import pandas as pd**

**#importing datasets**

**data\_set= pd.read\_csv(‘Dataset.csv’)**

**#Extracting Independent Variable**

**x= data\_set.iloc[:, :-1].values**

**#Extracting Dependent variable**

**y= data\_set.iloc[:, 3].values**

**#handling missing data(Replacing missing data with the mean value)**

**from sklearn.preprocessing import Imputer**

**imputer= Imputer(missing\_values =’NaN’, strategy=’mean’, axis = 0)**

**#Fitting imputer object to the independent varibles x.**

**imputerimputer= imputer.fit(x[:, 1:3])**

**#Replacing missing data with the calculated mean value**

**x[:, 1:3]= imputer.transform(x[:, 1:3])**

**#for Country Variable**

**from sklearn.preprocessing import LabelEncoder, OneHotEncoder**

**label\_encoder\_x= LabelEncoder()**

**x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])**

**#Encoding for dummy variables**

**onehot\_encoder= OneHotEncoder(categorical\_features= [0])**

**x= onehot\_encoder.fit\_transform(x).toarray()**

**#encoding for purchased variable**

**labelencoder\_y= LabelEncoder()**

**y= labelencoder\_y.fit\_transform(y)**

**# Splitting the dataset into training and test set.**

**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)**

**#Feature Scaling of datasets**

**from sklearn.preprocessing import StandardScaler**

**st\_x= StandardScaler()**

**x\_train= st\_x.fit\_transform(x\_train)**

**x\_test= st\_x.transform(x\_test)**