Tutorial 2

Data Preprocessing in Machine learning

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV **file**. However, sometimes, we may also need to use an HTML or xlsx file.

What is a CSV File?

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

Here we will use a demo dataset for data preprocessing, and for practice, it can be downloaded from here, "https://www.superdatascience.com/pages/machine-learning. For real-world problems, we can download datasets online from various sources such as https://www.kaggle.com/uciml/datasets, https://archive.ics.uci.edu/ml/index.php etc.

We can also create our dataset by gathering data using various API with Python and put that data into a .csv file.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

1. import numpy as nm

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

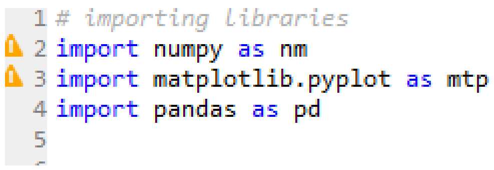
Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

1. import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Here, we have used pd as a short name for this library. Consider the below image:



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

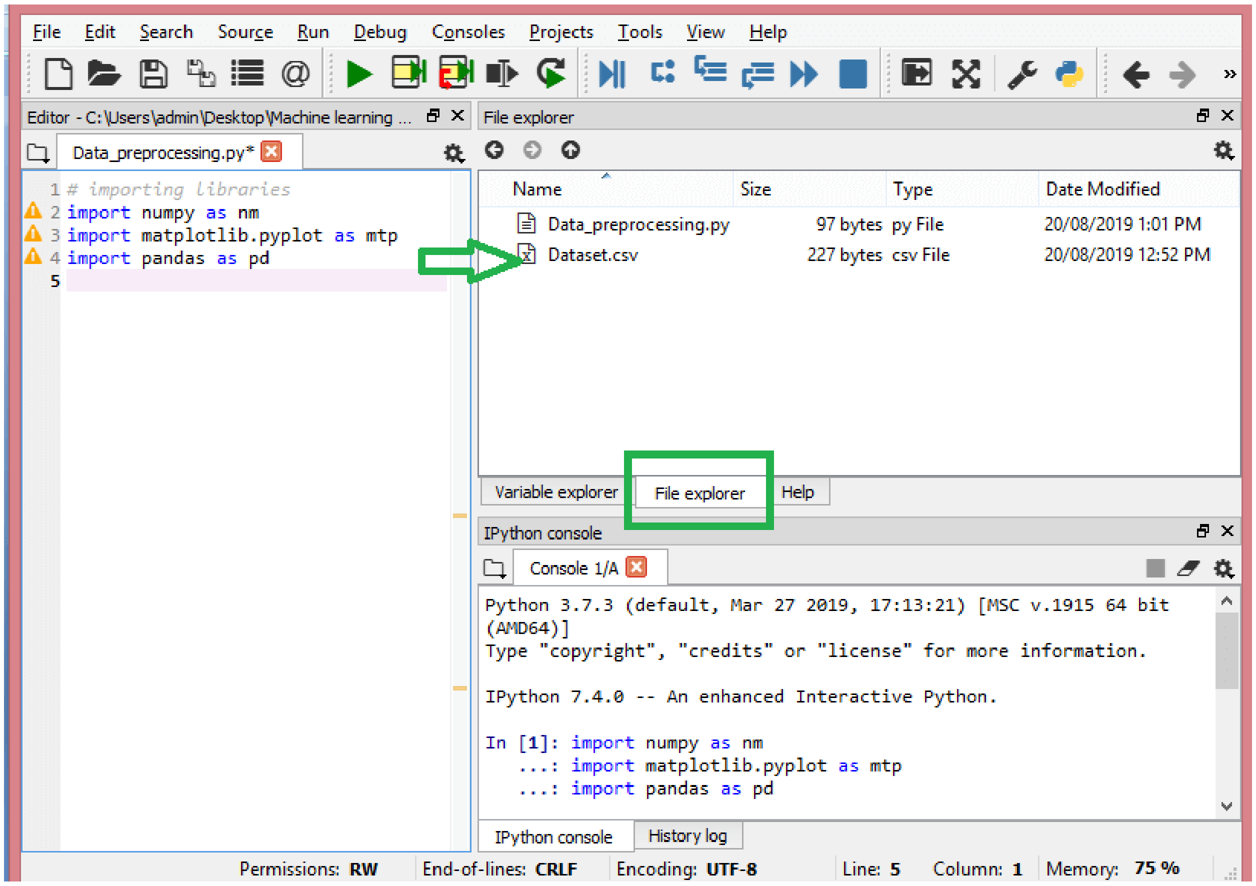
3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. Save your Python file in the directory which contains dataset.

1. Go to File explorer option in Spyder IDE, and select the required directory.
2. Click on F5 button or run option to execute the file.

Note: We can set any directory as a working directory, but it must contain the required dataset.

Here, in the below image, we can see the Python file along with required dataset. Now, the current folder is set as a working directory.



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
Name                   Size        Type      Date Modified
Data_preprocessing.py  97 bytes    py File   20/08/2019 1:01 PM
Dataset.csv            227 bytes   csv File  20/08/2019 12:52 PM
```

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: import numpy as nm
   ...: import matplotlib.pyplot as mtp
   ...: import pandas as pd
```

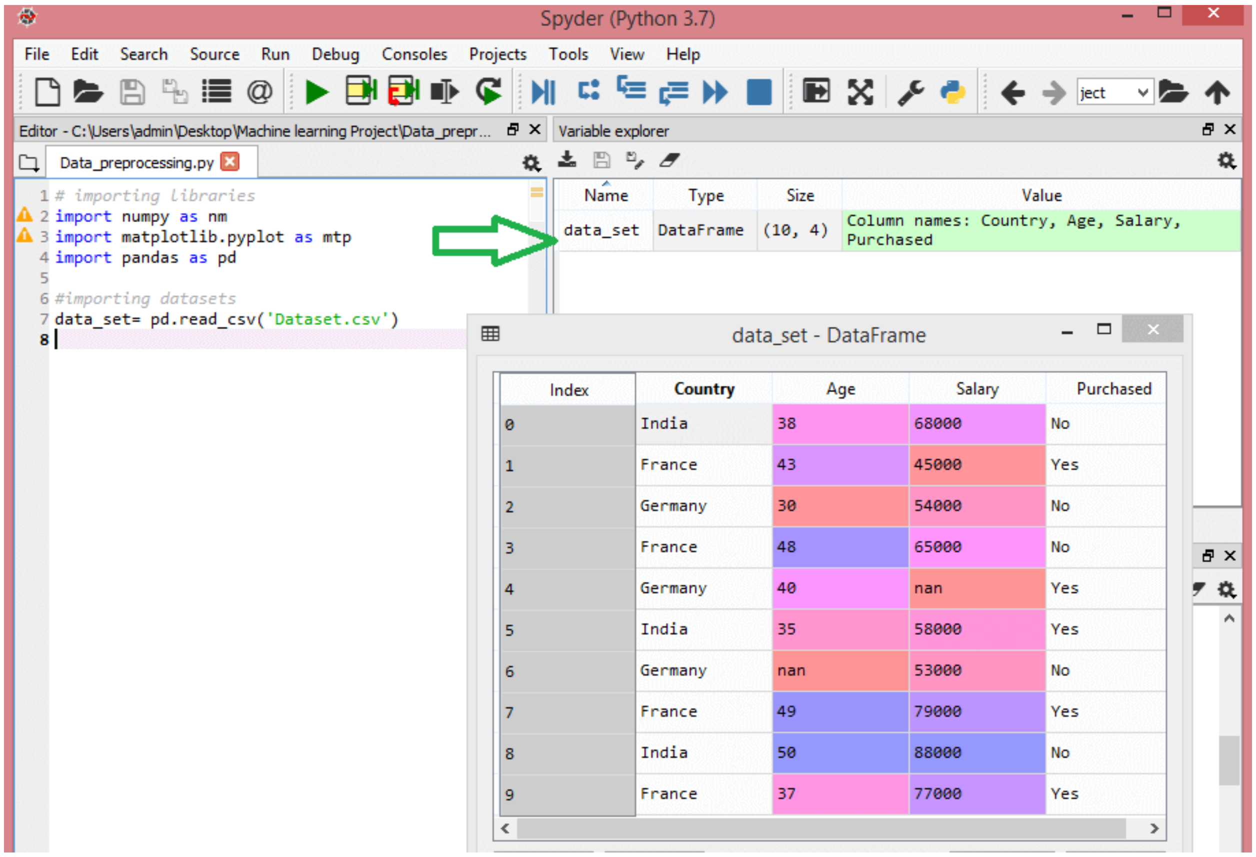
read_csv() function:

Now to import the dataset, we will use read_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use read_csv function as below:

1. data_set= pd.read_csv('Dataset.csv')

Here, **data_set** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on **data_set**. Consider the below image:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\admin\Desktop\Machine learning Project\Data_prepr...
Data_preprocessing.py
Variable explorer
data_set - DataFrame

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Dataset.csv')
```

Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased

Index	Country	Age	Salary	Purchased
0	India	38	68000	No
1	France	43	45000	Yes
2	Germany	30	54000	No
3	France	48	65000	No
4	Germany	40	nan	Yes
5	India	35	58000	Yes
6	Germany	nan	53000	No
7	France	49	79000	Yes
8	India	50	88000	No
9	France	37	77000	Yes

As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country, Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

Extracting independent variable:

To extract an independent variable, we will use **iloc[ ]** method of Pandas library. It is used to extract the required rows and columns from the dataset.

1. x= data_set.iloc[:,:-1].values

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

By executing the above code, we will get output as:

1. [['India' 38.0 68000.0]
2. ['France' 43.0 45000.0]
3. ['Germany' 30.0 54000.0]
4. ['France' 48.0 65000.0]
5. ['Germany' 40.0 nan]
6. ['India' 35.0 58000.0]
7. ['Germany' nan 53000.0]
8. ['France' 49.0 79000.0]
9. ['India' 50.0 88000.0]
10. ['France' 37.0 77000.0]]

As we can see in the above output, there are only three variables.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

1. y= data_set.iloc[:,3].values

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

By executing the above code, we will get output as:

Output:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

4) Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for it:

```
#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 variables x.
Imputer imputer= imputer.fit(x[:, 1:3])
#Replacing missing data with the calculated mean value
x[:, 1:3]= imputer.transform(x[:, 1:3])
```

Output:

```
array([['India', 38.0, 68000.0],
       ['France', 43.0, 45000.0],
       ['Germany', 30.0, 54000.0],
       ['France', 48.0, 65000.0],
       ['Germany', 40.0, 65222.22222222222],
       ['India', 35.0, 58000.0],
       ['Germany', 41.111111111111114, 53000.0],
       ['France', 49.0, 79000.0],
       ['India', 50.0, 88000.0],
       ['France', 37.0, 77000.0]], dtype=object
```

As we can see in the above output, the missing values have been replaced with the means of rest column values.

5) Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

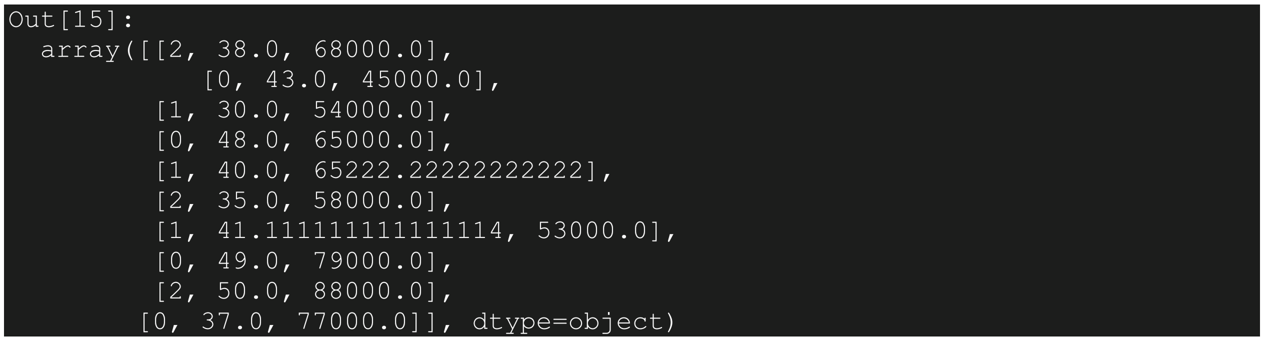
Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

```
#Catgorical data
#for Country Variable
from sklearn.preprocessing import LabelEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
```

Output:



```
Out[15]:
  array([[2, 38.0, 68000.0],
             [0, 43.0, 45000.0],
          [1, 30.0, 54000.0],
          [0, 48.0, 65000.0],
          [1, 40.0, 65222.22222222222],
          [2, 35.0, 58000.0],
          [1, 41.111111111111114, 53000.0],
          [0, 49.0, 79000.0],
          [2, 50.0, 88000.0],
         [0, 37.0, 77000.0]], dtype=object)
```

Explanation:

In above code, we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use **dummy encoding**.

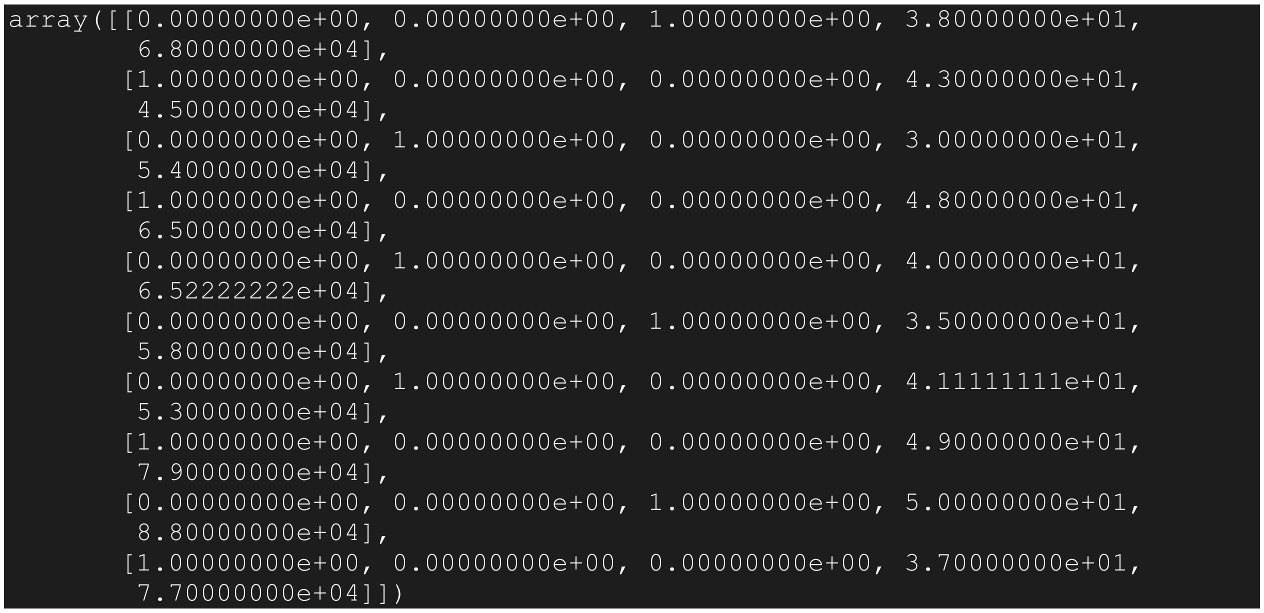
Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **preprocessing** library.

```
#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()
```

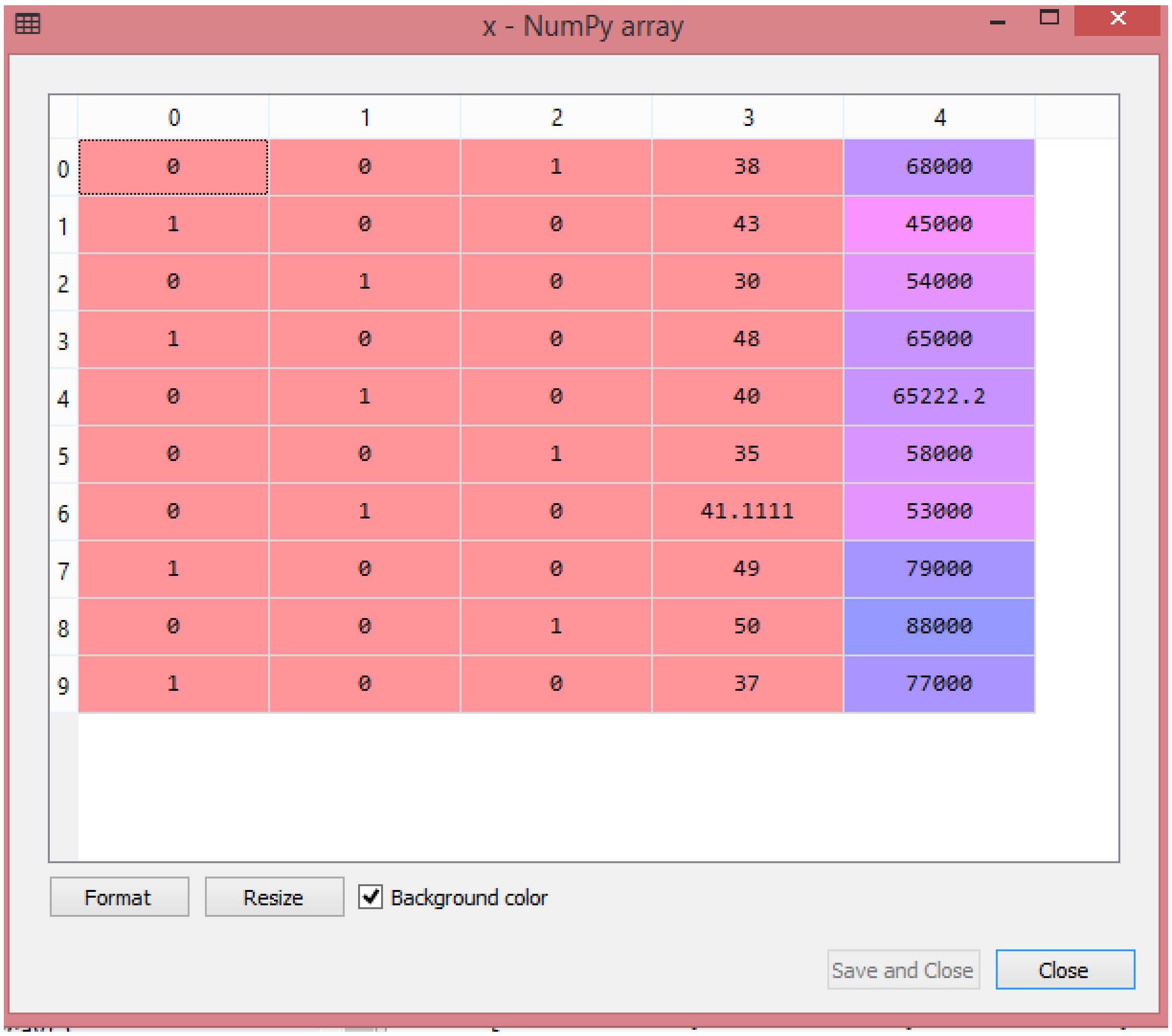
Output:



```
array([[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.30000000e+01,
        4.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        6.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.52222222e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.11111111e+01,
        5.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.90000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 5.00000000e+01,
        8.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        7.70000000e+04]])
```

As we can see in the above output, all the variables are encoded into numbers 0 and 1 and divided into three columns.

It can be seen more clearly in the variables explorer section, by clicking on x option as:



	0	1	2	3	4
0	0	0	1	38	68000
1	1	0	0	43	45000
2	0	1	0	30	54000
3	1	0	0	48	65000
4	0	1	0	40	65222.2
5	0	0	1	35	58000
6	0	1	0	41.1111	53000
7	1	0	0	49	79000
8	0	0	1	50	88000
9	1	0	0	37	77000

For Purchased Variable:

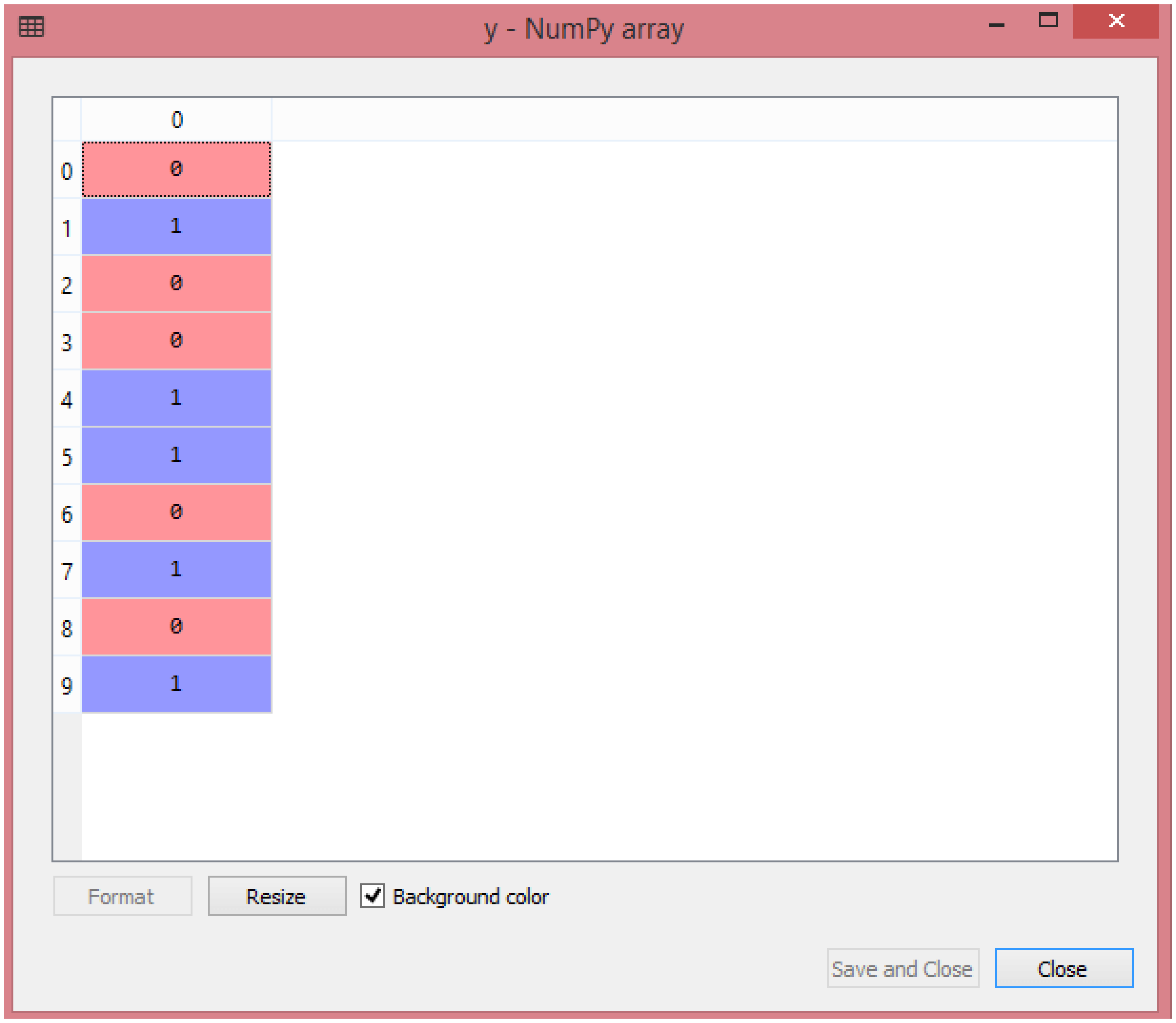
```
1. labelencoder_y= LabelEncoder()
2. y= labelencoder_y.fit_transform(y)
```

For the second categorical variable, we will only use labelencoder object of **LableEncoder** class. Here we are not using **OneHotEncoder** class because the purchased variable has only two categories yes or no, and which are automatically encoded into 0 and 1.

Output:

```
Out[17]: array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

It can also be seen as:



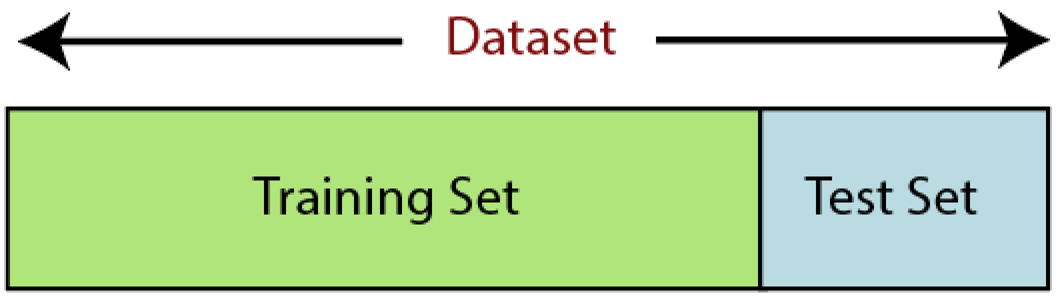
	0
0	0
1	1
2	0
3	0
4	1
5	1
6	0
7	1
8	0
9	1

6) Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

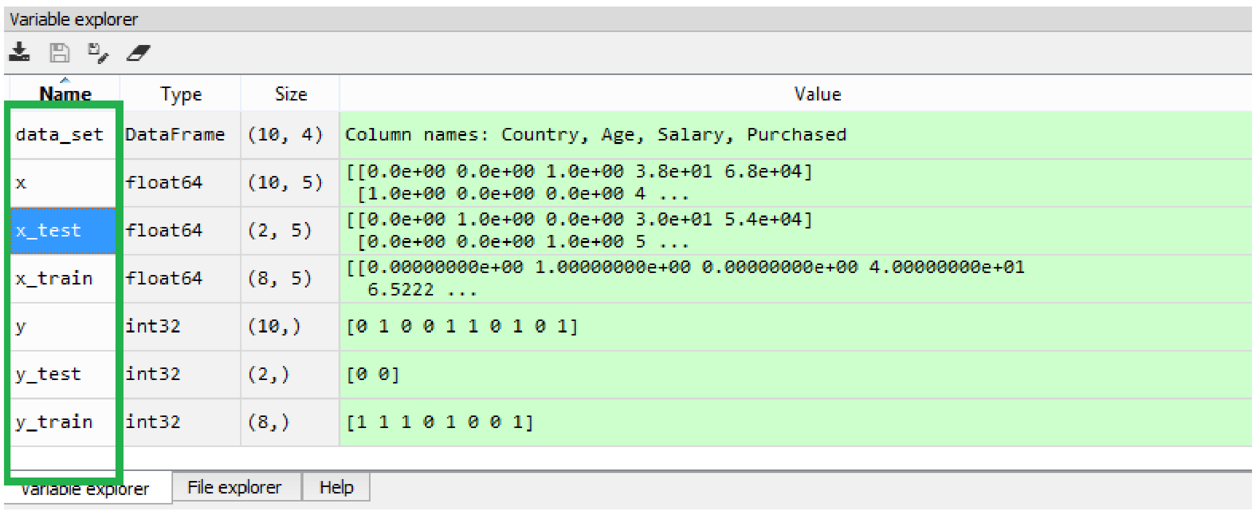
1. from sklearn.model_selection import train_test_split
2. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

Explanation:

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
  - **x_train:** features for the training data
  - **x_test:** features for testing data
  - **y_train:** Dependent variables for training data
  - **y_test:** Independent variable for testing data
- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter **random_state** is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

Output:

By executing the above code, we will get 4 different variables, which can be seen under the variable explorer section.



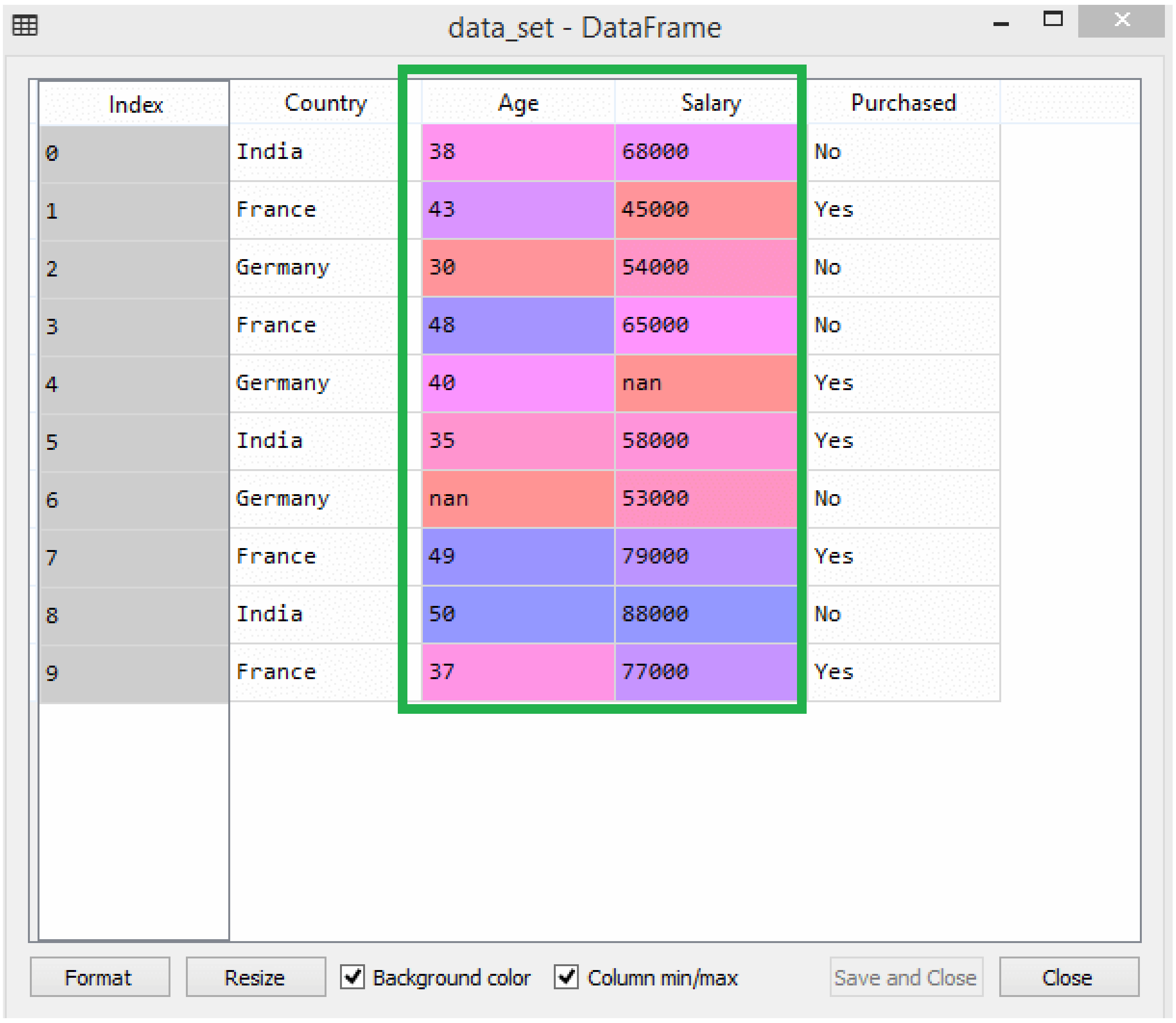
Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased
x	float64	(10, 5)	[[0.0e+00 0.0e+00 1.0e+00 3.8e+01 6.8e+04] [1.0e+00 0.0e+00 0.0e+00 4 ...
x_test	float64	(2, 5)	[[0.0e+00 1.0e+00 0.0e+00 3.0e+01 5.4e+04] [0.0e+00 0.0e+00 1.0e+00 5 ...
x_train	float64	(8, 5)	[[0.00000000e+00 1.00000000e+00 0.00000000e+00 4.00000000e+01 6.5222 ...
y	int32	(10,)	[0 1 0 0 1 1 0 1 0 1]
y_test	int32	(2,)	[0 0]
y_train	int32	(8,)	[1 1 1 0 1 0 0 1]

As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

7) Feature Scaling

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:



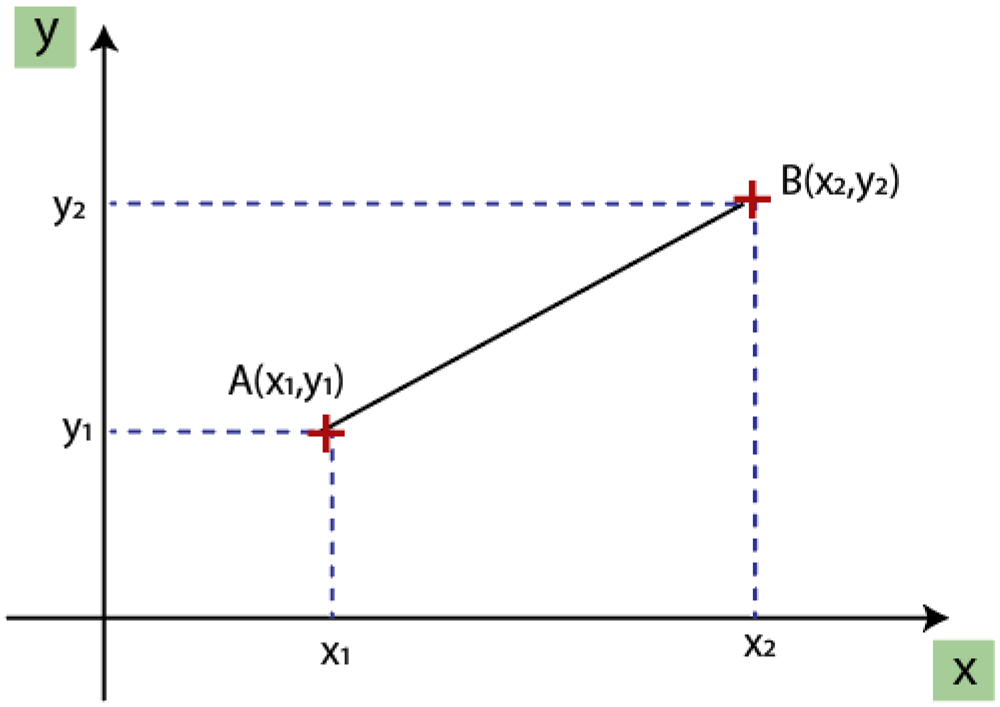
data_set - DataFrame

Index	Country	Age	Salary	Purchased
0	India	38	68000	No
1	France	43	45000	Yes
2	Germany	30	54000	No
3	France	48	65000	No
4	Germany	40	nan	Yes
5	India	35	58000	Yes
6	Germany	nan	53000	No
7	France	49	79000	Yes
8	India	50	88000	No
9	France	37	77000	Yes

As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it

will cause some issue in our machine learning model.

Euclidean distance is given as:

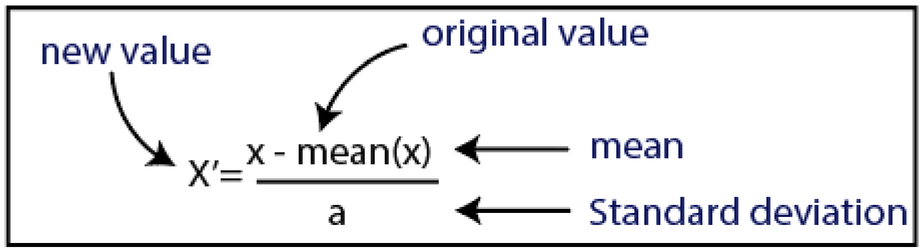


$$\text{Euclidean Distance Between A and B}=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

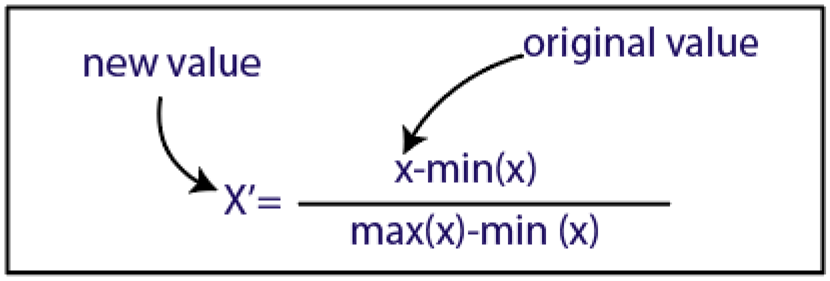
If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

There are two ways to perform feature scaling in machine learning:

Standardization



Normalization



Here, we will use the standardization method for our dataset.

For feature scaling, we will import ***StandardScaler*** class of ***sklearn.preprocessing*** library as:

```
1. from sklearn.preprocessing import StandardScaler
```

Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

```
1. st_x= StandardScaler()
2. x_train= st_x.fit_transform(x_train)
```

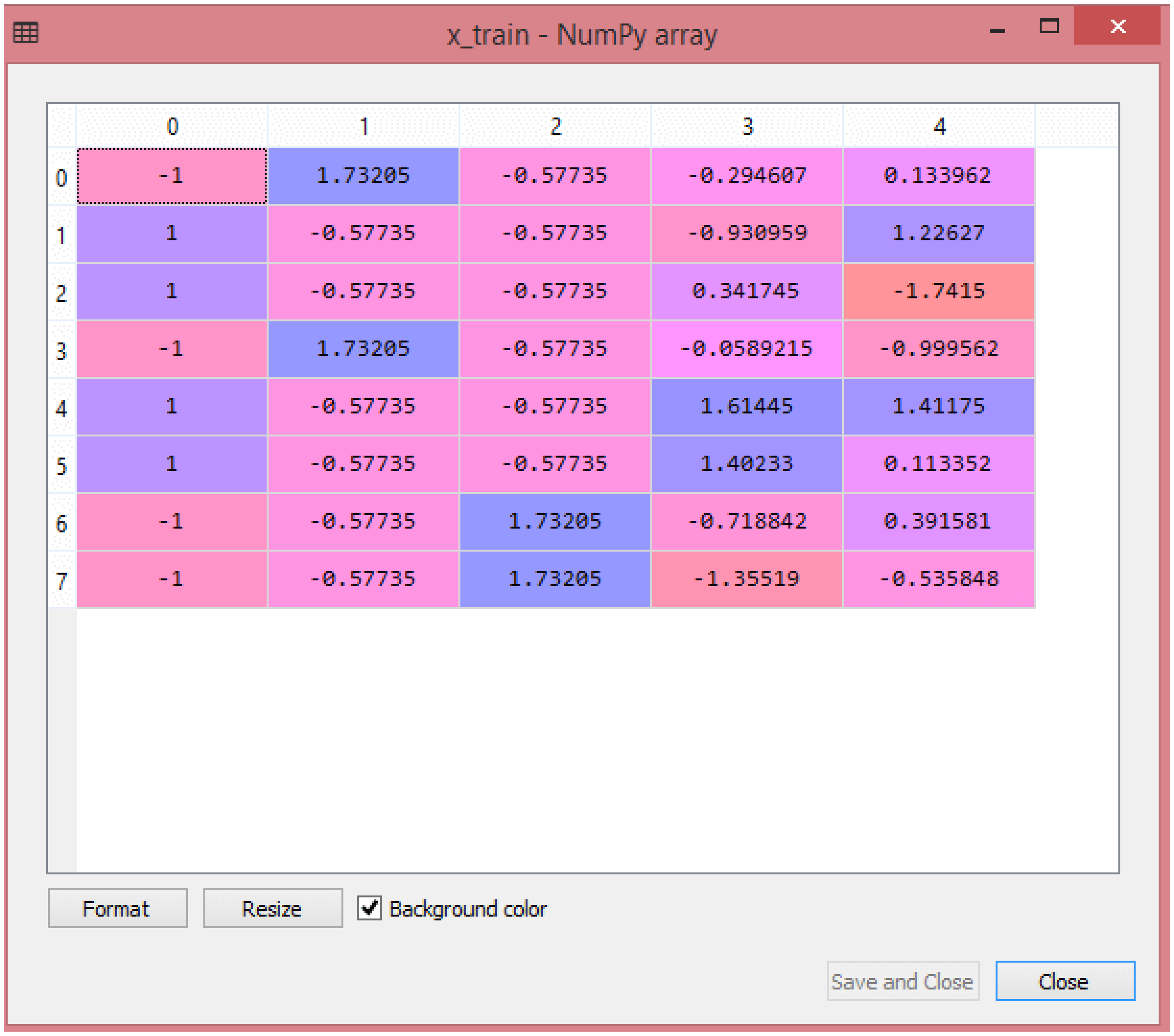
For test dataset, we will directly apply **transform()** function instead of **fit_transform()** because it is already done in training set.

```
1. x_test= st_x.transform(x_test)
```

Output:

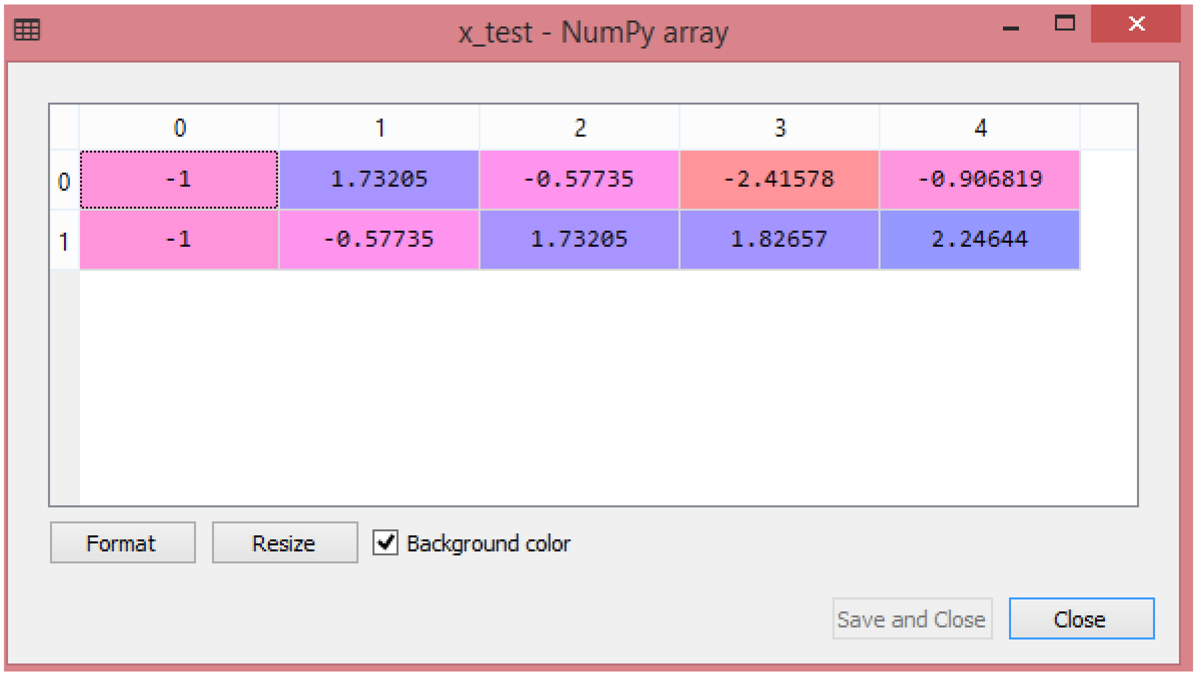
By executing the above lines of code, we will get the scaled values for x_train and x_test as:

x_train:


x_train - NumPy array

	0	1	2	3	4
0	-1	1.73205	-0.57735	-0.294607	0.133962
1	1	-0.57735	-0.57735	-0.930959	1.22627
2	1	-0.57735	-0.57735	0.341745	-1.7415
3	-1	1.73205	-0.57735	-0.0589215	-0.999562
4	1	-0.57735	-0.57735	1.61445	1.41175
5	1	-0.57735	-0.57735	1.40233	0.113352
6	-1	-0.57735	1.73205	-0.718842	0.391581
7	-1	-0.57735	1.73205	-1.35519	-0.535848

x_test:



	0	1	2	3	4
0	-1	1.73205	-0.57735	-2.41578	-0.906819
1	-1	-0.57735	1.73205	1.82657	2.24644

As we can see in the above output, all the variables are scaled between values -1 to 1.

Note: Here, we have not scaled the dependent variable because there are only two values 0 and 1. But if these variables will have more range of values, then we will also need to scale those variables.

Combining all the steps:

Now, in the end, we can combine all the steps together to make our complete code more understandable.

```
# 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)
```

In the above code, we have included all the data preprocessing steps together. But there are some steps or lines of code which are not necessary for all machine learning models. So we can exclude them from our code to make it reusable for all models.