**Predicting IMDb Scores Using Machine Learning**

TEAM MEMBER

732521104053 : THAMARAISELVI A

**Phase 3 Submission Document**

**Project :** Predicting IMDb Scores



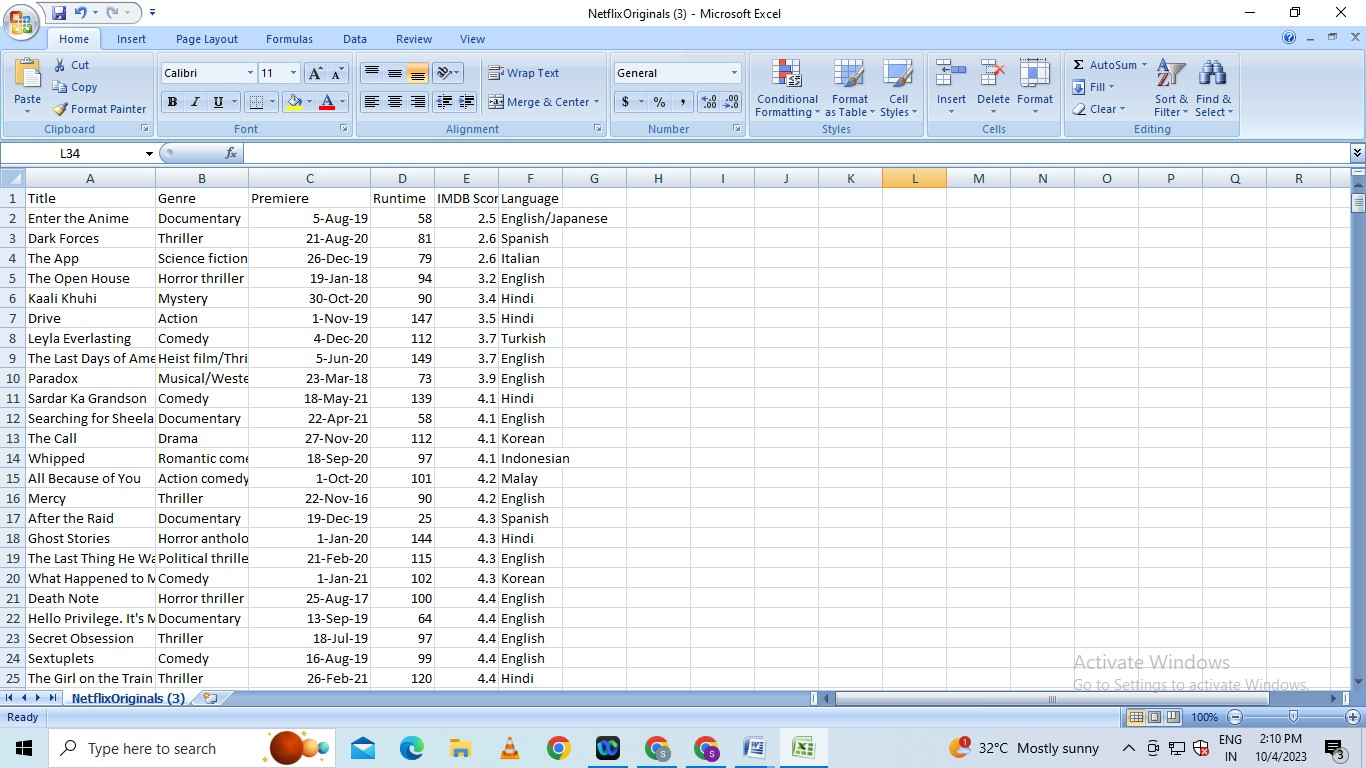
Introduction:

* Predicting IMDb scores for movies or TV shows typically involves using machine learning models and features such as cast, crew, genre, user reviews, and more. You can use regression algorithms to build a predictive model.
* The quality of your predictions depends on the quality and quantity of data, as well as the choice of features and model.
* In this project , we will explore advanced regression techniques to enhance the accuracy and robustness of IMDb scores prediction models
* Highlight the limitations of traditional linear regression models in capturing complex relationships.
* Emphasize the need for advanced regression techniques like Gradient Boosting and Neural Networks to enchance prediction accuracy.

**Content For Project Phase 2 :**

* this part you will begin building your project by loading and preprocessing the dataset.
* Begin building the IMDb score prediction model by loading and preprocessing the dataset.
* Load the movie dataset and preprocess the data for analysis.

**Data Source :**

* A Good Data for Predicting IMDb Scores using machine learning model should be Accurate , complete , accessible
* **Dataset Link : (**[**https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores**](https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores))

# DATA LOADING IN DATASET :

* Suppose if you want to start a ML project then what is the first and most important thing you would require? It is the data that we need to load for starting any of the ML project. With respect to data, the most common format of data for ML projects is CSV (comma-separated values).
* Basically, CSV is a simple file format which is used to store tabular data (number and text) such as a spreadsheet in plain text. In Python, we can load CSV data into with different ways but before loading CSV data we must have to take care about some considerations.

## Consideration While Loading CSV data :

* CSV data format is the most common format for ML data, but we need to take care about following major considerations while loading the same into our ML projects −

### File Header :

In CSV data files, the header contains the information for each field. We must use the same delimiter for the header file and for data file because it is the header file that specifies how should data fields be interpreted.

The following are the two cases related to CSV file header which must be considered −

* **Case-I: When Data file is having a file header** − It will automatically assign the names to each column of data if data file is having a file header.
* **Case-II: When Data file is not having a file header** − We need to assign the names to each column of data manually if data file is not having a file header.

In both the cases, we must need to specify explicitly weather our CSV file contains header or not.

### Comments :

* Comments in any data file are having their significance. In CSV data file, comments are indicated by a hash (#) at the start of the line. We need to consider comments while loading CSV data into ML projects because if we are having comments in the file then we may need to indicate, depends upon the method we choose for loading, whether to expect those comments or not.

### Delimiter :

* In CSV data files, comma (,) character is the standard delimiter. The role of delimiter is to separate the values in the fields. It is important to consider the role of delimiter while uploading the CSV file into ML projects because we can also use a different delimiter such as a tab or white space. But in the case of using a different delimiter than standard one, we must have to specify it explicitly.

### Quotes :

* In CSV data files, double quotation (“ ”) mark is the default quote character. It is important to consider the role of quotes while uploading the CSV file into ML projects because we can also use other quote character than double quotation mark. But in case of using a different quote character than standard one, we must have to specify it explicitly.

## Methods to Load CSV Data File

* While working with ML projects, the most crucial task is to load the data properly into it. The most common data format for ML projects is CSV and it comes in various flavors and varying difficulties to parse. In this section, we are going to discuss about three common approaches in Python to load CSV data file −

### Load CSV with Python Standard Library :

The first and most used approach to load CSV data file is the use of Python standard library which provides us a variety of built-in modules namely **csv module** and the reader()function. The following is an example of loading CSV data file with the help of it −

**Example :**

In this example, we are using the iris flower data set which can be downloaded into our local directory. After loading the data file, we can convert it into **NumPy** array and use it for ML projects. Following is the Python script for loading CSV data file −

First, we need to import the csv module provided by Python standard library as follows −

import csv

Next, we need to import Numpy module for converting the loaded data into NumPy array.

import numpy as np

Now, provide the full path of the file, stored on our local directory, having the CSV data file −

path = r"c:\iris.csv"

Next, use the csv.reader()function to read data from CSV file −

with open(path,'r') as f:

reader = csv.reader(f,delimiter = ',')

headers = next(reader)

data = list(reader)

data = np.array(data).astype(float)

We can print the names of the headers with the following line of script −

print(headers)

The following line of script will print the shape of the data i.e. number of rows & columns in the file −

print(data.shape)

Next script line will give the first three line of data file −print(data[:3])

**Output**

['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

(150, 4)

[ [5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

]

## Load CSV with NumPy

Another approach to load CSV data file is NumPy and numpy.loadtxt() function. The following is an example of loading CSV data file with the help of it −

### Example

In this example, we are using the Pima Indians Dataset having the data of diabetic patients. This dataset is a numeric dataset with no header. It can also be downloaded into our local directory. After loading the data file, we can convert it into NumPy array and use it for ML projects. The following is the Python script for loading CSV data file −

from numpy import loadtxt

path = r"C:\pima-indians-diabetes.csv"

datapath= open(path, 'r')

data = loadtxt(datapath, delimiter=",")

print(data.shape)

print(data[:3])

### Output

(768, 9)

[ [ 6. 148. 72. 35. 0. 33.6 0.627 50. 1.]

[ 1. 85. 66. 29. 0. 26.6 0.351 31. 0.]

[ 8. 183. 64. 0. 0. 23.3 0.672 32. 1.]

]

## Load CSV with Pandas

Another approach to load CSV data file is by **Pandas** and **pandas.read\_csv()function**. This is the very flexible function that returns a **pandas.DataFrame** which can be used immediately for plotting. The following is an example of loading CSV data file with the help of it −

### Example

Here, we will be implementing two Python scripts, first is with Iris data set having headers and another is by using the Pima Indians Dataset which is a numeric dataset with no header. Both the datasets can be downloaded into local directory.

**Script-1**

The following is the Python script for loading CSV data file using Pandas on Iris Data set −

from pandas import read\_csv

path = r"C:\iris.csv"

data = read\_csv(path)

print(data.shape)

print(data[:3])

Output:

(150, 4)

sepal\_length sepal\_width petal\_length petal\_width

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

**Script-2**

The following is the Python script for loading CSV data file, along with providing the headers names too, using Pandas on Pima Indians Diabetes dataset −

from pandas import read\_csv

path = r"C:\pima-indians-diabetes.csv"

headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names=headernames)

print(data.shape)

print(data[:3])

**Output**

(768, 9)

preg plas pres skin test mass pedi age class

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

The difference between above used three approaches for loading CSV data file can easily be understood with the help of given examples.

**gData Preprocessing in Machine learnin :**

* 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 BELOW GIVEN 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.

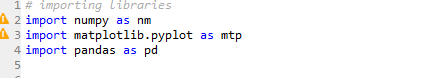
**Matplotlib:** The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

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:

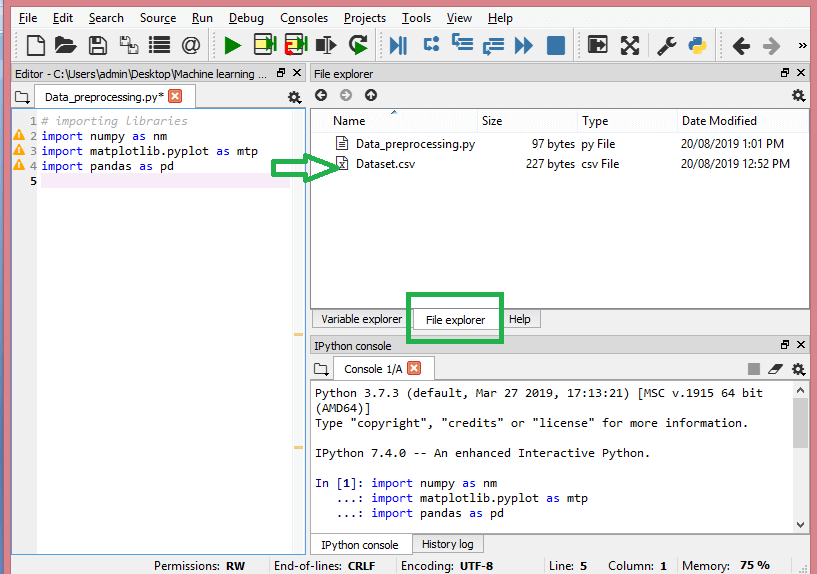
Here, we have used pd as a short name for this library. Consider the below image:



## 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. To set a working directory in Spyder IDE, we need to follow the below steps:

* Save your Python file in the directory which contains dataset.
* Go to File explorer option in Spyder IDE, and select the required directory.
* Click on F5 button or run option to execute the file.



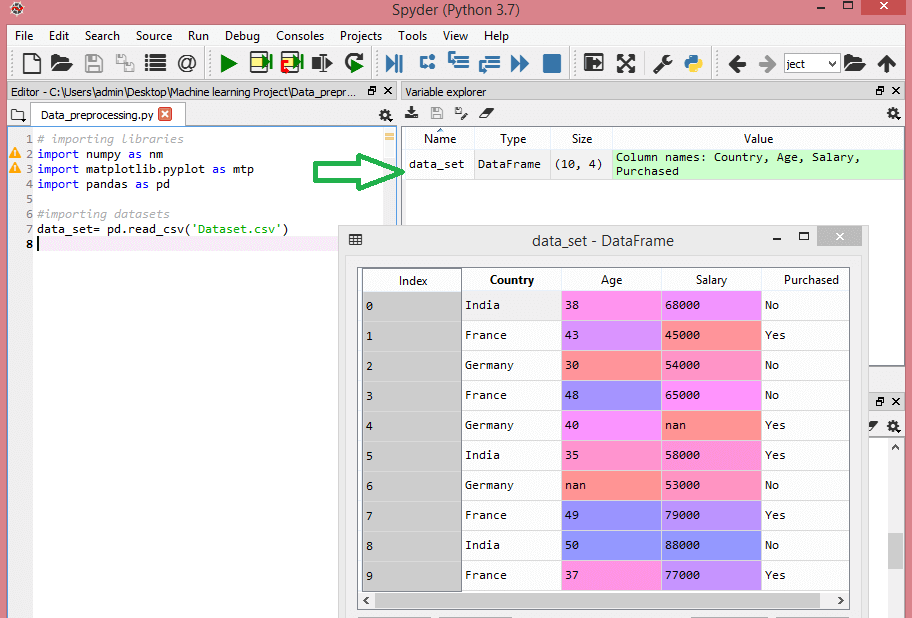
**read\_csv() function:**

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:



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)

#### Note: If you are using Python language for machine learning, then extraction is mandatory, but for R language it is not required.

## 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:

1. #handling missing data (Replacing missing data with the mean value)
2. from sklearn.preprocessing import Imputer
3. imputer= Imputer(missing\_values ='NaN', strategy='mean', axis = 0)
4. #Fitting imputer object to the independent variables x.
5. imputerimputer= imputer.fit(x[:, 1:3])
6. #Replacing missing data with the calculated mean value
7. 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.

1. #Catgorical data
2. #for Country Variable
3. from sklearn.preprocessing import LabelEncoder
4. label\_encoder\_x= LabelEncoder()
5. 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.

1. #for Country Variable
2. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
3. label\_encoder\_x= LabelEncoder()
4. x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])
5. #Encoding for dummy variables
6. onehot\_encoder= OneHotEncoder(categorical\_features= [0])
7. 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,

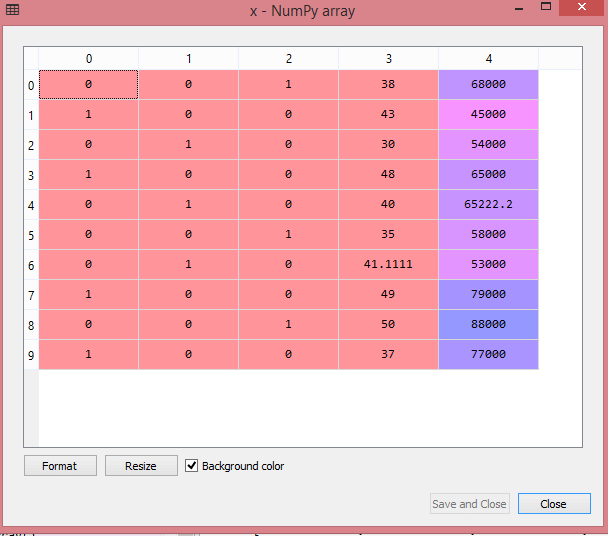
8.80000000e+04],

[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:



**For Purchased Variable:**

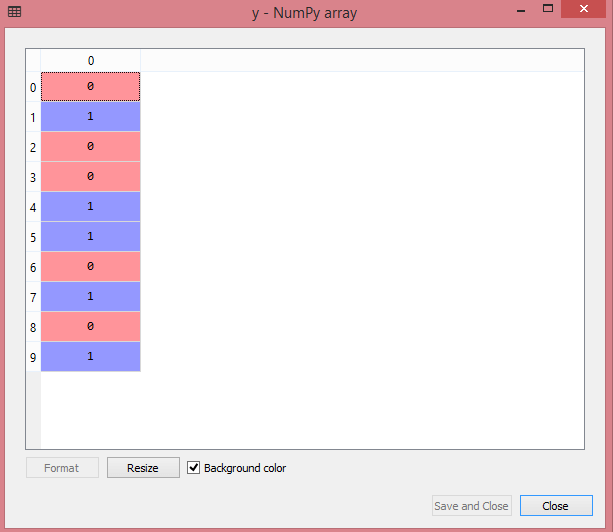
* labelencoder\_y= LabelEncoder()
* y= labelencoder\_y.fit\_transform(y)

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:**

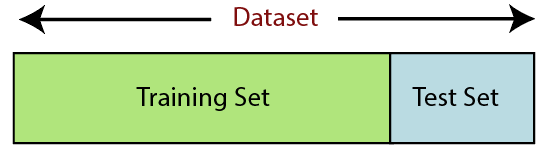


## 6) Splitting the Dataset into the Training set and Test set :

* 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.

* 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:

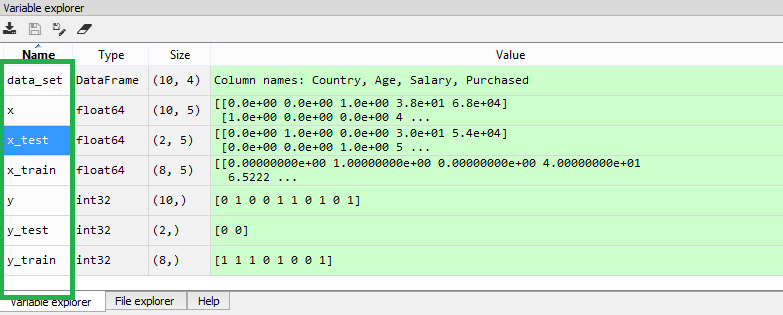
* from sklearn.model\_selection import train\_test\_split
* x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

**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.

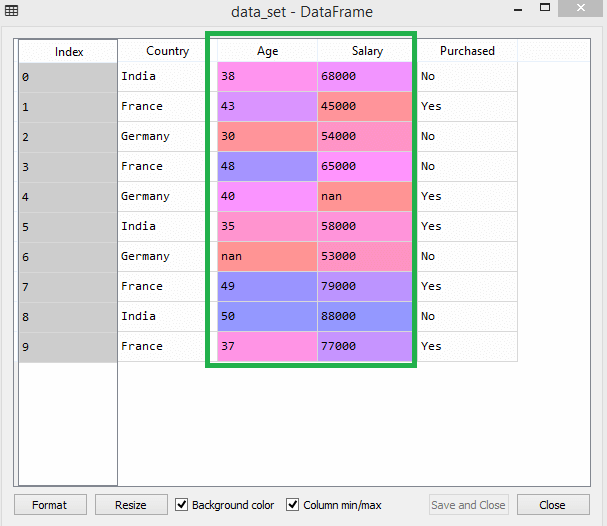


As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

## 7) Feature Scaling :

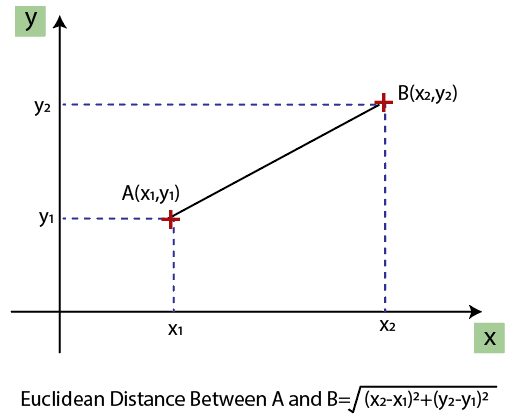
Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:



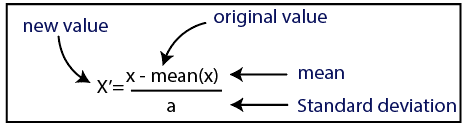
As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

Euclidean distance is given as:

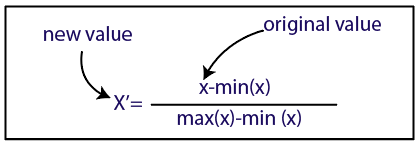


If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

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



**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)

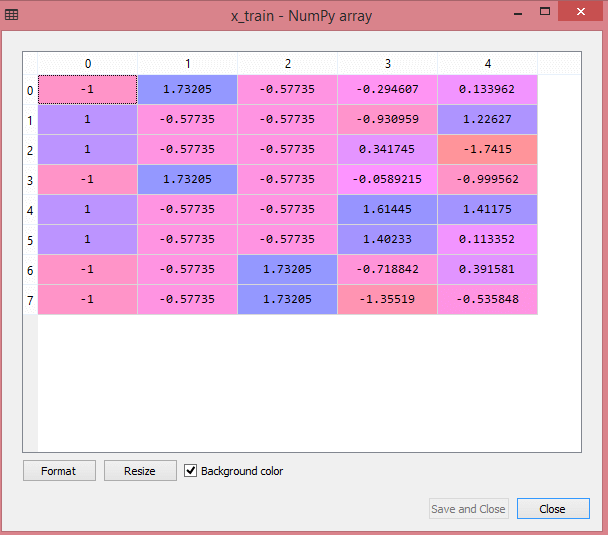
For test dataset, we will directly apply **transform()** function instead of **fit\_transform()** because it is already done in training set.

1. x\_test= st\_x.transform(x\_test)

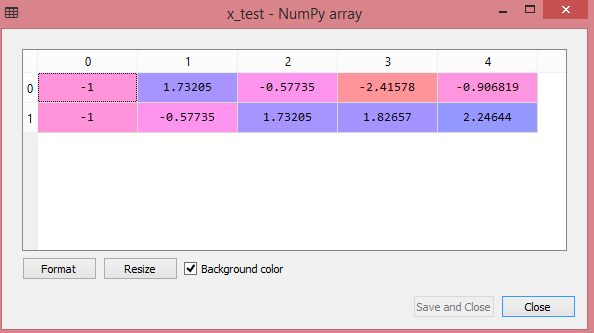
**Output:**

By executing the above lines of code, we will get the scaled values for x\_train and x\_test as:

**x\_train:**



**x\_test:**



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

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