Preparing Data

Data preparation (also referred to as “data preprocessing”) is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

Importance of Data Preparation:

Most machine learning algorithms require data to be formatted in a very specific way, so datasets generally require some amount of preparation before they can yield useful insights. Some datasets have values that are missing, invalid, or otherwise difficult for an algorithm to process. If data is missing, the algorithm can’t use it. If data is invalid, the algorithm produces less accurate or even misleading outcomes. Some datasets are relatively clean but need to be shaped (e.g., aggregated or pivoted) and many datasets are just lacking useful business context (e.g., poorly defined ID values), hence the need for feature enrichment. Good data preparation produces clean and well-curated data which leads to more practical, accurate model outcomes.

Hence, After selecting the raw data for ML training, the most important task is data pre-processing. In broad sense, data preprocessing will convert the selected data into a form we can work with or can feed to ML algorithms. We always need to preprocess our data so that it can be as per the expectation of machine learning algorithm.

Data Preprocessing Tools

Let’s talk about seven such techniques for Data Preprocessing in Python Machine Learning.

Rescaling Data:

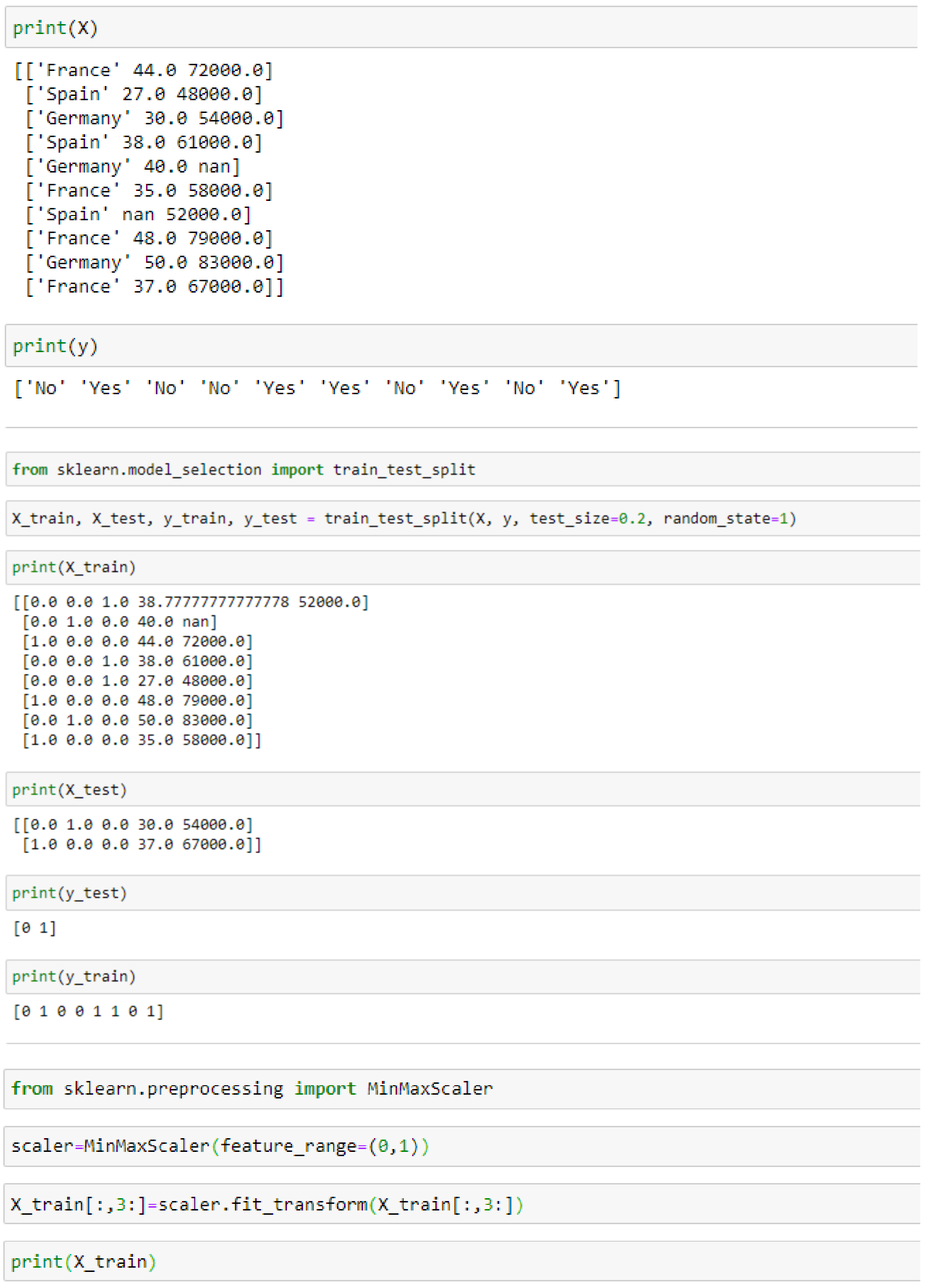
For data with attributes of varying scales, we can rescale attributes to possess the same scale. We rescale attributes into the range 0 to 1 and call it normalization. We use the MinMaxScaler class from scikit-learn. Let’s take an example

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

```python
dataset=pd.read_csv('Data.csv')
```

```python
dataset.head()
```

```
   Country  Age   Salary  Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
```



```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
print(y)
```

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

```
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=1)
```

```
print(X_train)
```

```
[[0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 40.0 nan]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

```
print(X_test)
```

```
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

```
print(y_test)
```

```
[0 1]
```

```
print(y_train)
```

```
[0 1 0 0 1 1 0 1]
```

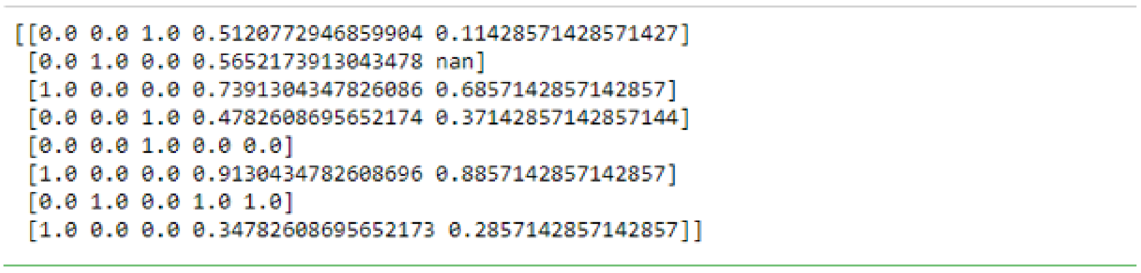
```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler=MinMaxScaler(feature_range=(0,1))
```

```
X_train[:,3:]=scaler.fit_transform(X_train[:,3:])
```

```
print(X_train)
```

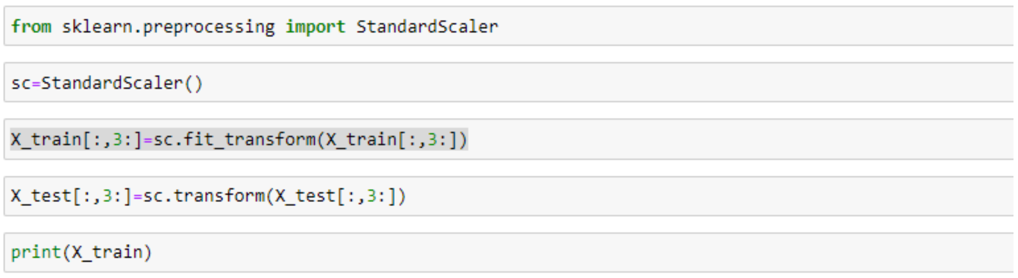
Hence on performing the data preprocessing using sckit package , we receive the following output in which all the features occur in between 0 and1 as follows:



```
[[0.0 0.0 1.0 0.5120772946859904 0.11428571428571427]
 [0.0 1.0 0.0 0.5652173913043478 nan]
 [1.0 0.0 0.0 0.7391304347826086 0.6857142857142857]
 [0.0 0.0 1.0 0.4782608695652174 0.37142857142857144]
 [0.0 0.0 1.0 0.0 0.0]
 [1.0 0.0 0.0 0.9130434782608696 0.8857142857142857]
 [0.0 1.0 0.0 1.0 1.0]
 [1.0 0.0 0.0 0.34782608695652173 0.2857142857142857]]
```

Standardizing Data

With standardizing, we can take attributes with a Gaussian distribution and different means and standard deviations and transform them into a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. For this, we use the StandardScaler class. Let’s continue with the same example.



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

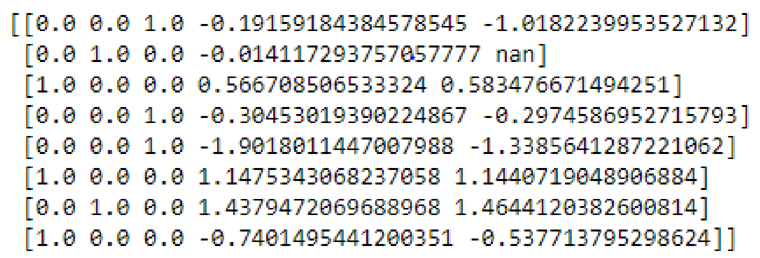
```
sc=StandardScaler()
```

```
X_train[:,3:]=sc.fit_transform(X_train[:,3:])
```

```
X_test[:,3:]=sc.transform(X_test[:,3:])
```

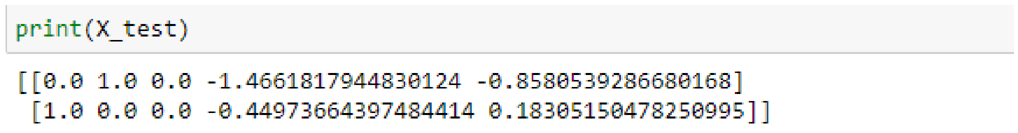
```
print(X_train)
```

The output on performing standard scaler on the X_train is as follows:



```
[[0.0 0.0 1.0 -0.19159184384578545 -1.0182239953527132]
 [0.0 1.0 0.0 -0.014117293757057777 nan]
 [1.0 0.0 0.0 0.566708506533324 0.583476671494251]
 [0.0 0.0 1.0 -0.30453019390224867 -0.2974586952715793]
 [0.0 0.0 1.0 -1.9018011447007988 -1.3385641287221062]
 [1.0 0.0 0.0 1.1475343068237058 1.1440719048906884]
 [0.0 1.0 0.0 1.4379472069688968 1.4644120382600814]
 [1.0 0.0 0.0 -0.7401495441200351 -0.537713795298624]]
```

The output for X_test scaling on printing is as follows:

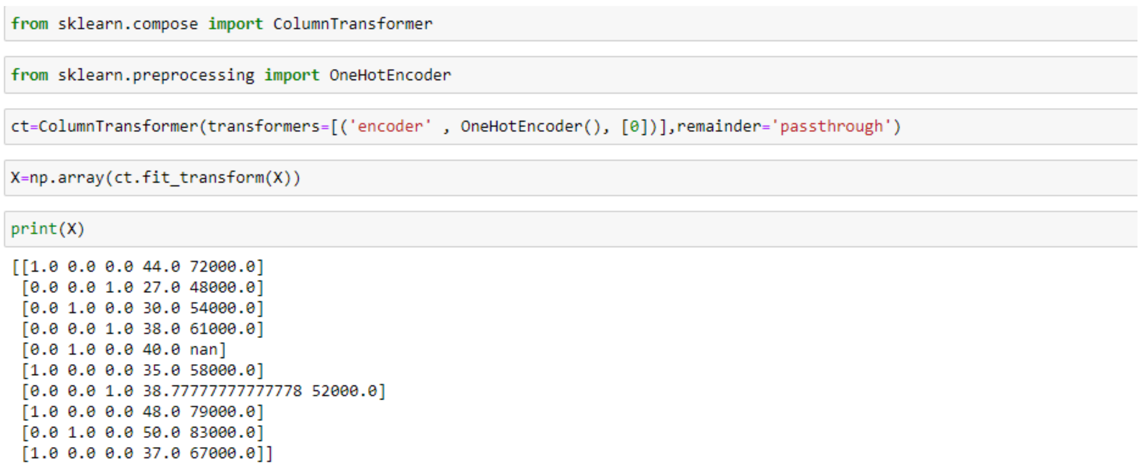


```
print(X_test)
```

```
[[0.0 1.0 0.0 -1.4661817944830124 -0.8580539286680168]
 [1.0 0.0 0.0 -0.44973664397484414 0.18305150478250995]]
```

OneHotEncoder

When dealing with few and scattered numerical values, we may not need to store these. Then, we can perform One Hot Encoding. Let’s continue with the same example by applying OneHotEncoder class imported from scikit Learn Package. Here we are only applying it to the country package to encode it in the form of o and 1.



```
from sklearn.compose import ColumnTransformer
```

```
from sklearn.preprocessing import OneHotEncoder
```

```
ct=ColumnTransformer(transformers=[('encoder' , OneHotEncoder(), [0])],remainder='passthrough')
```

```
X=np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 nan]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Hence in the above result we can see that the country names are encoded in the form of 0 and 1's, for example France is encoded as 1.0.0 .

Binarization

As the name suggests, this is the technique with the help of which we can make our data binary. We can use a binary threshold for making our data binary. The values above that threshold value will be converted to 1 and below that threshold will be converted to 0.

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded and then with the help of *Binarizer* class it will be converted into binary values i.e. 0 and 1 depending upon the threshold value. We are taking 0.5 as threshold value.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from sklearn.preprocessing import Binarizer
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Now, we can use class to convert the data into binary values.

```
binarizer = Binarizer(threshold=0.5).fit(array)
Data_binarized = binarizer.transform(array)
```

Here, we are showing the first 5 rows in the output.

```
print ("\nBinary data:\n", Data_binarized [0:5])
```

Output:

```
Binary data:
[[1. 1. 1. 1. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 0. 1. 0. 1. 0.]
[1.  1. 1. 0. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 1. 1. 0. 1. 0.]
[0.  1. 1. 1. 1. 1. 1. 1. 1.]]
```

Label Encoding

Most of the sklearn functions expect that the data with number labels rather than word labels. Hence, we need to convert such labels into number labels. This process is called label encoding. We can perform label encoding of data with the help of LabelEncoder() function of scikit-learn Python library.

n the following example, Python script will perform the label encoding.First, import the required Python libraries as follows –

```
import numpy as np
from sklearn import preprocessing
```

Now, we need to provide the input labels as follows –

```
input_labels =
['red','black','red','green','black','yellow','white']
```

The next line of code will create the label encoder and train it.

```
test_labels = ['green','red','black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
encoded_values = [3,0,4,1]
decoded_list = encoder.inverse_transform(encoded_values)
```

We can get the list of encoded values with the help of following python script –

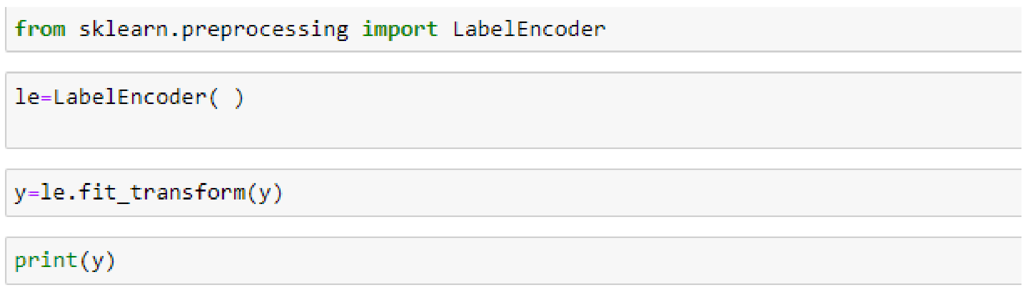
```
print("\nEncoded values =", encoded_values)
print("\nDecoded labels =", list(decoded_list)
```

Output:

```
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]
Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
```

Also, if we apply label encoding on Data.csv dataset we get the following:

Some labels can be words or numbers. Usually, training data is labelled with words to make it readable. Label encoding converts word labels into numbers to let algorithms work on them. Let's continue with the same example.



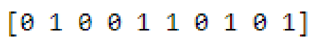
```
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder( )
```

```
y=le.fit_transform(y)
```

```
print(y)
```

In the above code we performed label encoding on y i.e the Purchased column to encode it in the form of 0 and 1.The following output shows the transformation:



```
[0 1 0 0 1 1 0 1 0 1]
```