Introduction

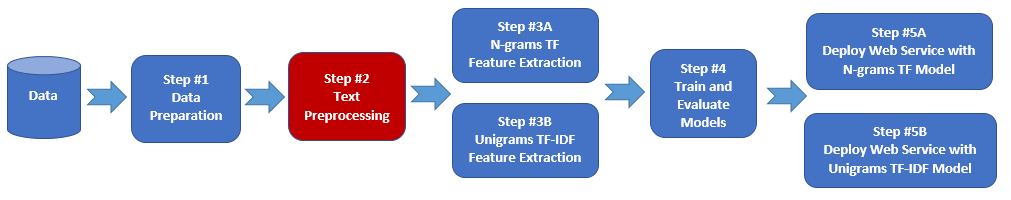
In this article, we are going to focused on data pre-processing techniques in python. We will be learning on many algorithms models that will make our data perform incredibly well. The term pre-processing refers to the transformations applied to your data before feeding it to the algorithm models. Thanks to scikit-learn library in python, which comes with a pre-built functionality under sklearn.preprocessing. It’s always a clever idea to prepare your data in such way to best expose the structure of the problem to the machine learning algorithms that you intend to use.

After finishing this article, you will be fully equipped with the basic techniques of data pre-processing. You will also discover how to prepare your data for machine learning in Python using scikit-learn.

We’ll follow these steps throughout this article.

1. Load the datasets
2. Impute the missing data
3. Apply a preprocessing transform to the categorical variables
4. Apply One Hot Encoder and Label Encoder
5. Apply Feature scaling
6. Split the dataset into train and test sets.

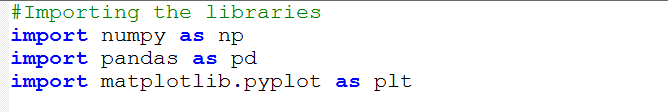
Let’s get started.



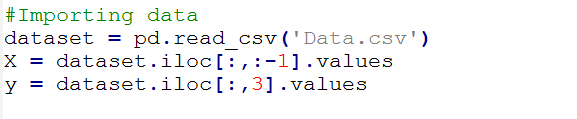
1 Load datasets

For this article, I have used a random data that I generate for this exercise, you can get it from here: [Download data](https://github.com/Olow304/Olow_Data_Machine_Learning/tree/master/Data%20Preprocessing%20Python)

Now, let’s get started by importing packages and the data set.



Data set,



Let’s take a closer look at our data set.

>>Print(dataset)

Country Age Salary Purchased

France 44.0 72000.0 No

Germany 30.0 54000.0 No

Spain 38.0 61000.0 No

Spain NaN 52000.0 No

Germany 50.0 83000.0 No

Spain 27.0 48000.0 Yes

Germany 40.0 NaN Yes

France 35.0 58000.0 Yes

France 48.0 79000.0 Yes

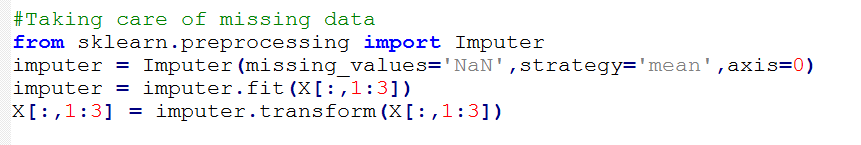
France 37.0 67000.0 Yes

2 Impute the missing data

There are many libraries out there for normalizing your data, such as [StandardScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html), [Normalizer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html), [Binarizer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Binarizer.html), and [Imputer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer) library.

For Imputer preprocessing can be useful for sparse datasets with attributes of varying scales when using algorithms that weight input values. Imputation is good for completing missing values.

You can impute your data in Python with scikit-learn to use the [Imputer](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer) class.



After imputing missing data, we’ll have something like this: without missing values

Country Age Salary Purchased

France 44.0 72000.0 No

Germany 30.0 54000.0 No

Spain 38.0 61000.0 No

Spain 38.7 52000.0 No

Germany 50.0 83000.0 No

Spain 27.0 48000.0 Yes

Germany 40.0 63777.7 Yes

France 35.0 58000.0 Yes

France 48.0 79000.0 Yes

France 37.0 67000.0 Yes

Red indicates imputing the values after applying with Imputer class

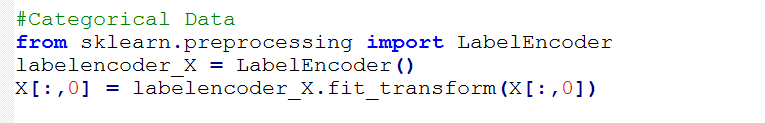
3 Applying Label Encoder and One Hot Encoder

Scikit-Learn provides us very good classes to handle categorical data. It’s very better than writing your custom function, you should use [LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) which is specially designed to handle categorical data.

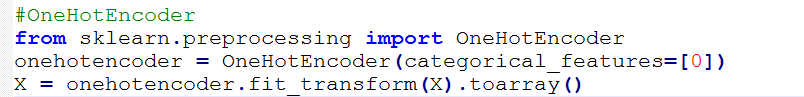
In our data we have four columns, Country, Age, Salary, Purchased. Our **Country** column contains categorical data which we’ll going to apply to LabelEncoder and as well as [OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html).

One-Hot Encoding transforms each categorical feature with n possible values into n binary features, with only one active.

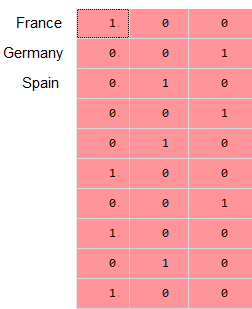
You can LabelEncoder your data in Python with scikit-learn to use the [LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) class.



You can [OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) your data in Python with scikit-learn to use the [OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) class.



After applying [LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) and [OneHotEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) into our data, we’ll have the following output:

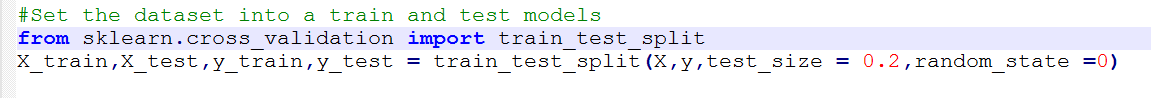
What it is saying is that, if you look at the OneHotEncoder output. We have three columns, the first column represents ‘France’, second column is ‘Spain and the last column is ‘Germany’. In our Country column, we have the first variable ‘France’ so we’ll put ‘1’ for ‘France’ in our OneHotEncoder and 0, 0 for the rest columns, because the first row has ‘France’ in it, etc. [Helpful link](https://www.youtube.com/watch?v=iZ3e_cifP7Y)

4 Split the dataset into train and test sets

We can take our original dataset, split it into two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results.

The size of the split can depend on the size and specifics of your dataset, although it is common to use 80% of the data for training and the remaining 20% for testing.

We can use [cross\_validation](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) class in sklearn library to accomplish this task.



4 Applying Feature Scaling

Feature scaling is the method to limit the range of variables so that they can be compared on common grounds. It is performed on continuous variables.

In our data, we have continuous values for ‘Age’ and ‘Salary’ if we were to input into our machine learning model without applying to feature scaling, we’ll not get an accurate prediction. In that case, we will need to make our data balanced in terms of scaling.

We’ll be using, Standardization which is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

you can standardize data using scikit-learn with the [StandardScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) class.

