**Balance the dataset**

The data looks imbalanced as the three categories from the Category column of the data are not equally distributed like:

IT - 8377

Marketing and Communication - 5227

Sales – 2919

There are various methods to balance the data, but there are some advantages and disadvantages of using them, so after analysing few methods, we can drill down to one statistical method which is **Stratified k-fold cross-validation** method, where we can split a dataset randomly, although in such a way that maintains the same class distribution in each subset and target variable, the class, is used to control the sampling process.

**Stratified k-fold** cross-validation will enforce the class distribution in each split of the data to match the distribution in the complete training dataset and results in skill estimates that generally have a lower bias than other methods.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=5 becoming 5-fold cross-validation.

The value of k is usually chosen between 5 – 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller.

The scikit-learn library provides an implementation that will split a given data sample up.

**sklearn.model\_selection.StratifiedKFold.**

The StratifiedKFold() scikit-learn class can be used. It takes as arguments the number of splits, whether or not to shuffle the sample, and the seed for the pseudorandom number generator used prior to the shuffle.

**kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=1)**

The split() function can then be called on the class where the data sample is provided as an argument. Called repeatedly, the split will return each group of train and test sets. Specifically, arrays are returned containing the indexes into the original data sample of observations to use for train and test sets on each iteration.

**for train, test in kfold.split(data):**

**print('train: %s, test: %s' % (train, test))**

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split or normal k-folds.

## Feature Extraction

## Selecting the model

## Evaluating the model