**Handling missing data:**

There are two ways by which we can handle missing values in our dataset. The first method is commonly used to handle null values. Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 75% of missing values. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias.

In the second method, we replace all the NaN values with either mean, median, or most frequent values. This is an approximation that can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to the removal of rows and columns. Replacing the above three approximations is a statistical approach to handling the missing values. This method is also called **leaking the data** while training.

For dealing with missing data, we will use the [Imputer](https://scikit-learn.org/stable/modules/impute.html?source=post_page---------------------------) library from sklearn.preprocessing package. Instead of providing mean, you can also provide median or most frequent value in the strategy parameter.

from sklearn.preprocessing import Imputer

imputer = Imputer(missing\_values='NaN', strategy = 'mean', axis = 0)

The next step is to train the imputer instance with the data stored in X(predictors).

imputer = imputer.fit(X[:,1:3])

X[:, 1:3] = imputer.transform(X[:,1:3])

**Encoding the categorical data:**

Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set.

Some examples include:

A “pet” variable with the values: “dog” and “cat”.

A “color” variable with the values: “red”, “green”, and “blue”.

A “place” variable with the values: “first”, “second”, and “third”.

*Each value represents a different category.*

### Note: What is the Problem with Categorical Data?

Some algorithms can work with categorical data directly. But many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves. This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

We are going to use a technique called label encoding. Label encoding is simply converting each value in a column to a number. For example, the body\_style column contains 5 different values. We could choose to encode it like this:

* convertible -> 0
* hardtop -> 1
* hatchback -> 2
* sedan -> 3
* wagon -> 4

To implement Label encoding we will import LabelEncoder from sklearn.preprocessing package. But it labels categories as 0,1,2,3…. Now since 0<1<2, the equations in your regression model may think one category has a higher value than the other, which is of course not true.

To solve this situation we have a concept called [Dummy variables](https://en.wikipedia.org/wiki/Dummy_variable_(statistics)?source=post_page---------------------------). In regression analysis, a dummy variable is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. They are used as devices to sort data into mutually exclusive categories (such as smoker/non-smoker, etc.).

To implement the concept of dummy variables we will import [OneHotEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html?source=post_page---------------------------) library from sklearn.preprocessing package. You need to provide the column index which needs to be encoded under categorical\_features. So if a column has 3 categories, 3 columns will be created and likewise for any number of categories.

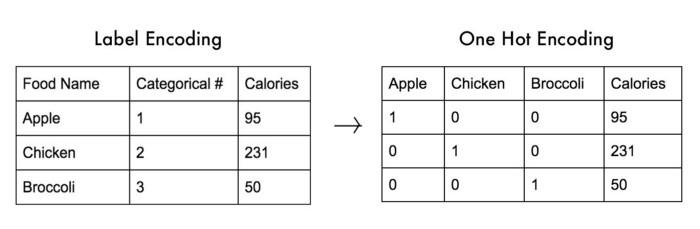
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder\_X = LabelEncoder()

X[:,0] = labelencoder\_X.fit\_transform(X[:,0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

****