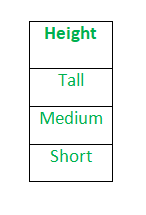
**Label Encoding** refers to converting the labels into a numeric form so as to convert them into the machine-readable form.

**Example :**   
Suppose we have a column*Height* in some dataset.



After applying label encoding, the Height column is converted into:

Chart

Description automatically generated with medium confidence

where 0 is the label for tall, 1 is the label for medium, and 2 is a label for short height.

We apply *Label Encoding* on iris dataset on the target column which is Species. It contains three species *Iris-setosa, Iris-versicolor, Iris-virginica*

**Limitation of label Encoding**   
Label encoding converts the data in machine-readable form, but it assigns a unique number(starting from 0) to each class of data. This may lead to the generation of priority issues in the training of data sets. A label with a high value may be considered to have high priority than a label having a lower value.

**One Hot Encoding:**

We use this categorical data encoding technique when the features are nominal(do not have any order). In one hot encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.

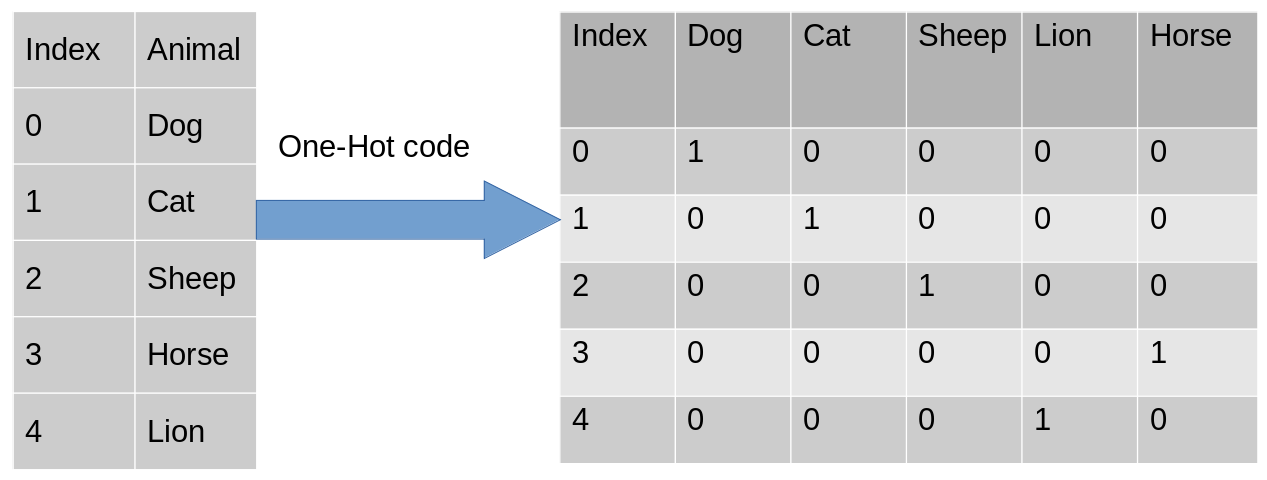
In this technique, the categorical parameters will prepare separate columns for both Male and Female labels. So, wherever there is Male, the value will be 1 in Male column and 0 in Female column, and vice-versa. Let’s understand with an example: Consider the data where fruits and their corresponding categorical values and prices are given.

| Fruit | Categorical value of fruit | Price |
| --- | --- | --- |
| apple | 1 | 5 |
| mango | 2 | 10 |
| apple | 1 | 15 |
| orange | 3 | 20 |

The output after one-hot encoding of the data is given as follows,

| apple | mango | orange | price |
| --- | --- | --- | --- |
| 1 | 0 | 0 | 5 |
| 0 | 1 | 0 | 10 |
| 1 | 0 | 0 | 15 |
| 0 | 0 | 1 | 20 |

These newly created binary features are known as**Dummy variables.** The number of dummy variables depends on the levels present in the categorical variable. This might sound complicated. Let us take an example to understand this better. Suppose we have a dataset with a category animal, having different animals like Dog, Cat, Sheep, Cow, Lion. Now we have to one-hot encode this data.



After encoding, in the second table, we have dummy variables each representing a category in the feature Animal. Now for each category that is present

## **Binary Encoding**

Binary encoding is a combination of Hash encoding and one-hot encoding. In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns.

Binary encoding works really well when there are a high number of categories. For example the cities in a country where a company supplies its products.

#Import the libraries

import category\_encoders as ce

import pandas as pd

#Create the Dataframe

data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad','Mumbai','Agra']})

#Create object for binary encoding

encoder= ce.BinaryEncoder(cols=['city'],return\_df=True)

#Original Data

data

Graphical user interface, text, application

Description automatically generated

#Fit and Transform Data

data\_encoded=encoder.fit\_transform(data)

data\_encoded

A picture containing text

Description automatically generated

Binary encoding is a memory-efficient encoding scheme as it uses fewer features than one-hot encoding. Further, It reduces the curse of dimensionality for data with high cardinality.

## **Target Encoding**

Target encoding is a Baysian encoding technique.

Bayesian encoders use information from dependent/target variables to encode the categorical data.

In target encoding, we calculate the mean of the target variable for each category and replace the category variable with the mean value. In the case of the categorical target variables, the posterior probability of the target replaces each category..

#import the libraries

import pandas as pd

import category\_encoders as ce

#Create the Dataframe

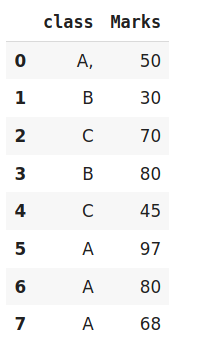
data=pd.DataFrame({'class':['A,','B','C','B','C','A','A','A'],'Marks':[50,30,70,80,45,97,80,68]})

#Create target encoding object

encoder=ce.TargetEncoder(cols='class')

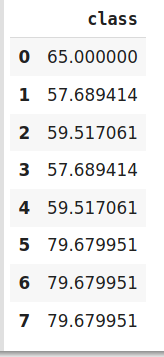
#Original Data

Data



#Fit and Transform Train Data

encoder.fit\_transform(data['class'],data['Marks'])



We perform Target encoding for train data only and code the test data using results obtained from the training dataset. Although, a very efficient coding system, it has the following **issues** responsible for deteriorating the model performance-

1. It can lead to target leakage or overfitting. To address overfitting we can use different techniques.
   1. In the leave one out encoding, the current target value is reduced from the overall mean of the target to avoid leakage.
   2. In another method, we may introduce some Gaussian noise in the target statistics. The value of this noise is hyperparameter to the model.
2. The second issue, we may face is the improper distribution of categories in train and test data. In such a case, the categories may assume extreme values. Therefore the target means for the category are mixed with the marginal mean of the target.

**Ordinal Encoding: —**Where Order of data matters for example:

* low, medium, high
* cold, hot, lava Hot

|  |
| --- |
| * **from** sklearn.preprocessing **import** OrdinalEncoder * ord1 **=** OrdinalEncoder() * # fitting encoder * ord1.fit([df['ord\_2']]) * # transforming the column after fitting * df["ord\_2"] **=** ord1.transform(df[["ord\_2"]]) * df.head(10) |

**Output:**

Table

Description automatically generated

**Frequency Encoding:**We can also encode considering the frequency distribution. This method can be effective at times for nominal features.

**Code:**

|  |
| --- |
| # grouping by frequency  fq **=** df.groupby('nom\_0').size()**/**len(df)  # mapping values to dataframe  df.loc[:, "{}\_freq\_encode".format('nom\_0')] **=** df['nom\_0'].map(fq)  # drop original column.  df **=** df.drop(['nom\_0'], axis**=**1)  fq.plot.bar(stacked**=**True)  df.head(10) |

**Output:**

Chart, bar chart

Description automatically generated

*Frequency distribution (fq)*

Table

Description automatically generated

*Output*