Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

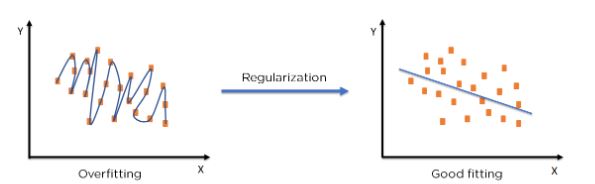
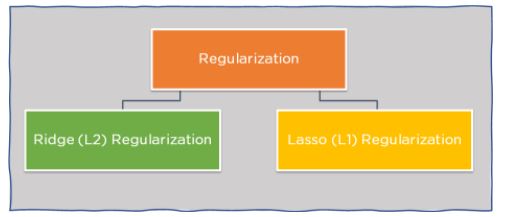


                                                  Figure 5: Regularization on an over-fitted model

Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

**Regularization Techniques**

There are two main types of regularization techniques: Ridge Regularization and Lasso Regularization.



The term regularization in the context of machine learning refers to a collection of strategies that help the machine learn more than merely memorize.

Let’s imagine you’re conducting a cat vs. dog classification and your trained model achieves 98 percent accuracy on training data, but only 82 percent accuracy on test data — your model is remembering rather than generalizing.

* In summary, if your model’s assessment metrics for the training and testing datasets diverge significantly, it’s considered to have an overfitting problem.

## Regularization

Regularization refers to a range of strategies for regularizing learning from specific characteristics in classical algorithms or neurons in neural network algorithms.

It normalizes and moderates weights associated with a feature or a neuron so that algorithms aren’t reliant on a small number of features or neurons to predict the outcome. This method aids in avoiding the issue of overfitting.

* This method reduces the computational cost of a complicated model by converting it to a simpler one to avoid overfitting.

## Regularization Algorithms

* **Ridge regression –** Its purpose is to overcome problems such as data overfitting and multicollinearity in data. When there is considerable collinearity (the existence of near-linear connections among the independent variables) among the feature variables, a typical linear or polynomial regression model will fail. Ridge Regression adjusts the variables by a modest squared bias factor. The feature variable coefficients are pulled away from this rigidity by such a squared bias factor, providing a little bit of bias into the model but considerably lowering variation.

## Ridge is an excellent way to prevent overfitting.

Use regularization to solve overfitting and feature selection if you have a model with a high number of features in the dataset and want to prevent making the model too complicated.

However, the ridge has one major drawback: the final model has all N characteristics.

Ridge regression decreases the two coefficients towards each other when the variables are highly linked. Lasso is torn between the two and prefers one over the other.

One never knows which variable will be chosen depending on the situation. Elastic-net is a hybrid of the two that tries to shrink while still doing the sparse selection.

* **LASSO** – It simply penalizes large coefficients, in contrast to Ridge Regression. When the hyperparameter is big enough, Lasso has the effect of driving certain coefficient estimations to be absolutely zero. As a result, Lasso conducts variable selection, resulting in models that are significantly easier to read than Ridge Regression models. In a nutshell, it’s about lowering variability and increasing the accuracy of linear regression models.

If we have a large number of features, LASSO works effectively for feature selection.

It reduces coefficients to zero and if a set of predictors is highly associated, lasso selects one and reduces the others to zero.

**Scaling**

Feature engineering is a critical step in building accurate and effective [machine learning models](https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/). One key aspect of feature engineering is scaling, normalization, and standardization, which involves transforming the data to make it more suitable for modeling. These techniques can help to improve model performance, reduce the impact of outliers, and ensure that the data is on the same scale.

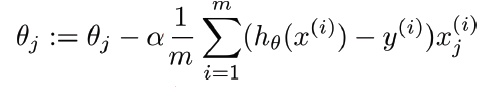
## What is Feature Scaling?

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process

#### 1. Gradient Descent Based Algorithms

**Machine learning algorithms like** [**linear regression**](https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/)**,** [**logistic regression**](https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/)**,** [**neural network**](https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/)**, PCA (principal component analysis), etc., that use gradient descent as an optimization technique require data to be scaled.** Take a look at the formula for gradient descent below:

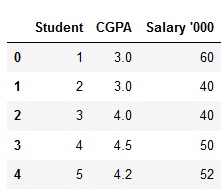
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/gradient-descent.png)

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in the ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

#### 2. Distance-Based Algorithms

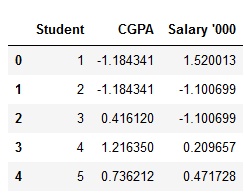
Distance algorithms like [KNN](https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), [K-means clustering](https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), and [SVM](https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/)(support vector machines) are most affected by the range of features. This is because, behind the scenes, **they are using distances between data points to determine their similarity.**

For example, let’s say we have data containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/knn_ex.png)

Since both the features have different scales, there is a chance that higher weightage is given to features with higher magnitudes. This will impact the performance of the machine learning algorithm; obviously, we do not want our algorithm to be biased towards one feature.

Therefore, we scale our data before employing a distance based algorithm so that all the features contribute equally to the result.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/knn_ex_scaled.png)

The effect of scaling is conspicuous when we compare the Euclidean distance between data points for students A and B, and between B and C, before and after scaling, as shown below:

* Distance AB before scaling =>Euclidean distance
* Distance BC before scaling =>Euclidean distance
* Distance AB after scaling =>Euclidean distance
* Distance BC after scaling =>Euclidean distance

#### 3. Tree-Based Algorithms

[Tree-based algorithms](https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature.

So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

## What is Normalization?

Normalization is a data preprocessing technique used to adjust the values of features in a dataset to a common scale. This is done to facilitate data analysis and modeling, and to reduce the impact of different scales on the accuracy of machine learning models.

**Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.**

Here’s the formula for normalization:

[Normalization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Norm_eq.gif)

Here, Xmax and Xmin are the maximum and the minimum values of the feature, respectively.

* When the value of X is the minimum value in the column, the numerator will be 0, and hence X’ is 0
* On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator, and thus the value of X’ is 1
* If the value of X is between the minimum and the maximum value, then the value of X’ is between 0 and 1

## What is Standardization?

Standardization is another scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Here’s the formula for standardization:

[Standardization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Stand_eq.gif)  
Feature scaling: Muis the mean of the feature values and Feature scaling: Sigmais the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization? Let’s find out!

## The Big Question – Normalize or Standardize?

| **Normalization** | **Standardization** |
| --- | --- |
| Rescales values to a range between 0 and 1 | Centers data around the mean and scales to a standard deviation of 1 |
| Useful when the distribution of the data is unknown or not Gaussian | Useful when the distribution of the data is Gaussian or unknown |
| Sensitive to outliers | Less sensitive to outliers |
| Retains the shape of the original distribution | Changes the shape of the original distribution |
| May not preserve the relationships between the data points | Preserves the relationships between the data points |
| Equation: (x – min)/(max – min) | Equation: (x – mean)/standard deviation |

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized, and standardized data and comparing the performance for the best results.**

It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required.

**Data Encoding algorithms**

**Data Encoding** is an important pre-processing step in Machine Learning. It refers to the process of converting categorical or textual data into numerical format, so that it can be used as input for algorithms to process. The reason for encoding is that most machine learning algorithms work with numbers and not with text or categorical variables.

The choice of encoding method can have a significant impact on model performance, so it is important to choose an appropriate encoding technique based on the nature of the data and the specific requirements of the model.

**There are several methods for encoding categorical variables, including**

1. One-Hot Encoding

2. Dummy Encoding

3.Ordinal Encoding

4. Binary Encoding

5. Count Encoding

6. Target Encoding

Let’s take a closer look at each of these methods.

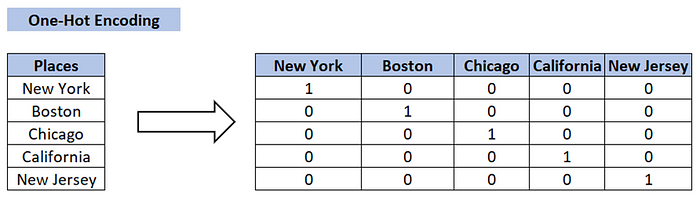
# One-Hot Encoding:

• One-Hot Encoding is the **Most Common** method for encoding **Categorical** variables.

• a **Binary Column** is created for each **Unique Category** in the variable.

• If a category is present in a sample, the corresponding column is set to 1, and all other columns are set to 0.

• For example, if a variable has three categories ‘A’, ‘B’ and ‘C’, three columns will be created and a sample with category ‘B’ will have the value [0,1,0].



# One-Hot Encoding:   
# create a sample dataframe with a categorical variable  
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red']})  
  
# perform one-hot encoding on the 'color' column  
one\_hot = pd.get\_dummies(df['color'])  
  
# concatenate the one-hot encoding with the original dataframe  
df1 = pd.concat([df, one\_hot], axis=1)  
  
# drop the original 'color' column  
df1 = df1.drop('color', axis=1)

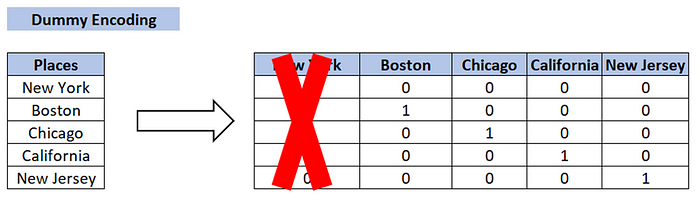
# Dummy Encoding

• Dummy coding scheme is **similar to one-hot encoding**.

• This categorical data encoding method transforms the categorical variable into a set of binary variables [0/1].

• In the case of **one-hot encoding**, for N categories in a variable, it uses N binary variables.

• The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N labels/categories.



One-Hot Encoding vs Dummy Encoding:

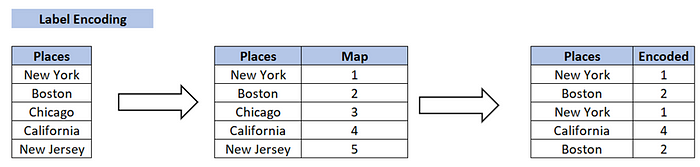
**One-Hot Encoding** — N categories in a variable, **N** binary variables.

**Dummy encoding** — N categories in a variable, **N-1** binary variables.

# Create a sample dataframe with categorical variable  
data = {'Color': ['Red', 'Green', 'Blue', 'Red', 'Blue']}  
df = pd.DataFrame(data)  
  
# Use get\_dummies() function for dummy encoding  
dummy\_df = pd.get\_dummies(df['Color'], drop\_first=True, prefix='Color')  
  
# Concatenate the dummy dataframe with the original dataframe  
df = pd.concat([df, dummy\_df], axis=1)

# **Label Encoding:**

* Each unique category is assigned a **Unique Integer** value.
* This is a simpler encoding method, but it has a **Drawback** in that the assigned integers may be **misinterpreted** by the machine learning algorithm **as having an Ordered Relationship** when in fact they **do not**.



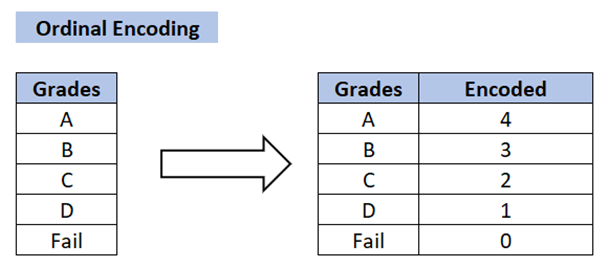
from sklearn.preprocessing import LabelEncoder  
  
# Create a sample dataframe with categorical data  
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green']})  
  
print(f"Before Encoding the Data:\n\n{df}\n")  
  
# Create a LabelEncoder object  
le = LabelEncoder()  
  
# Fit and transform the categorical data  
df['color\_label'] = le.fit\_transform(df['color'])

# Ordinal Encoding:

• Ordinal Encoding is used when the **categories** in a variable have a **Natural Ordering**.

• In this method, the **categories are assigned a numerical value** based on their order, such as 1, 2, 3, etc.

For example, if a variable has categories **‘Low’, ‘Medium’ and ‘High’**, they can be assigned the values **1, 2, and 3**, respectively.

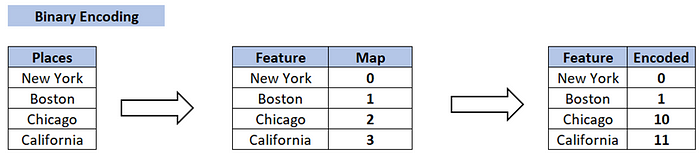


# Ordinal Encoding:  
# create a sample dataframe with a categorical variable  
df = pd.DataFrame({'quality': ['low', 'medium', 'high', 'medium']})  
print(f"Before Encoding the Data:\n\n{df}\n")  
  
# specify the order of the categories  
quality\_map = {'low': 0, 'medium': 1, 'high': 2}  
  
# perform ordinal encoding on the 'quality' column  
df['quality\_map'] = df['quality'].map(quality\_map)

# Binary Encoding:

• Binary Encoding is **similar** to **One-Hot Encoding**, but **instead of creating a separate column** for each category, the categories are represented as **binary digits**.

* For example, if a variable has four categories ‘A’, ‘B’, ‘C’ and ‘D’, they can be represented as 0001, 0010, 0100 and 1000, respectively.

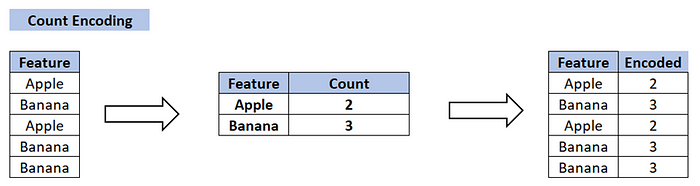


# Binary Encoding:  
  
import pandas as pd  
  
# create a sample dataframe with a categorical variable  
df = pd.DataFrame({'animal': ['cat', 'dog', 'bird', 'cat']})  
print(f"Before Encoding the Data:\n\n{df}\n")  
  
# perform binary encoding on the 'animal' column  
animal\_map = {'cat': 0, 'dog': 1, 'bird': 2}  
df['animal'] = df['animal'].map(animal\_map)  
df['animal'] = df['animal'].apply(lambda x: format(x, 'b'))  
  
# print the resulting dataframe  
print(f"After Encoding the Data:\n\n{df}\n")

# Count Encoding:

• Count Encoding is a method for encoding categorical variables by **counting the number of times a category appears** in the dataset.

* For example, if a variable has categories ‘A’, ‘B’ and ‘C’ and category ‘A’ appears 10 times in the dataset, it will be assigned a value of 10.



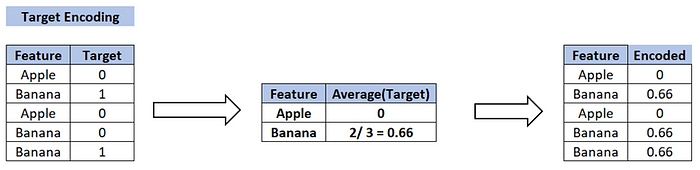
# Count Encoding:  
# create a sample dataframe with a categorical variable  
df = pd.DataFrame({'fruit': ['apple', 'banana', 'apple', 'banana']})  
print(f"Before Encoding the Data:\n\n{df}\n")  
  
# perform count encoding on the 'fruit' column  
counts = df['fruit'].value\_counts()  
df['fruit'] = df['fruit'].map(counts)  
  
# print the resulting dataframe  
print(f"After Encoding the Data:\n\n{df}\n")

**Target Encoding:**

• This is a more **advanced encoding technique** used for dealing with **high cardinality categorical features**, i.e., features with many unique categories.

• The average target value for each category is calculated and this average value is used to replace the categorical feature.

* This has the **advantage of considering the relationship between the target and the categorical feature**, but it can also **lead to overfitting** if not used with caution.



## Effect Encoding

This encoding technique is also known as **Deviation Encoding** or **Sum Encoding.** Effect encoding is almost similar to dummy encoding, with a little difference. In dummy coding, we use 0 and 1 to represent the data but in effect encoding, we use three values i.e. 1,0, and -1.

The row containing only 0s in dummy encoding is encoded as -1 in effect encoding.  In the dummy encoding example, the city Bangalore at index 4  was encoded as 0000. Whereas in effect encoding it is represented by -1-1-1-1.

Let us see how we implement it in python-

import category\_encoders as ce

import pandas as pd

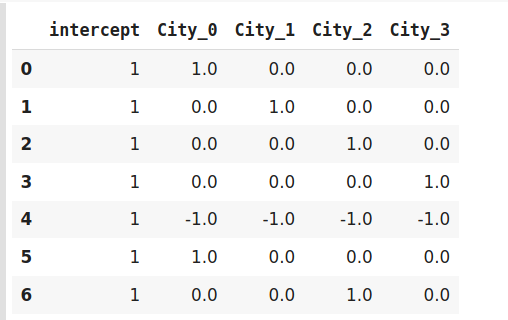
data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi,'Hyderabad']}) encoder=ce.sum\_coding.SumEncoder(cols='City',verbose=False,)

#Original Data

data



encoder.fit\_transform(data)



Effect encoding is an advanced technique. In case you are interested to know more about effect encoding, refer to [this](https://www.researchgate.net/publication/256349393_Categorical_Variables_in_Regression_Analysis_A_Comparison_of_Dummy_and_Effect_Coding) interesting paper.

## Hash Encoder

To understand Hash encoding it is necessary to know about hashing. Hashing is the transformation of arbitrary size input in the form of a fixed-size value. We use hashing algorithms to perform hashing operations i.e to generate the hash value of an input. Further, hashing is a one-way process, in other words, one can not generate original input from the hash representation.

Hashing has several applications like data retrieval, checking data corruption, and in data encryption also. We have multiple hash functions available for example Message Digest (MD, MD2, MD5), Secure Hash Function (SHA0, SHA1, SHA2), and many more.

Just like one-hot encoding, the Hash encoder represents categorical features using the new dimensions. Here, the user can fix the number of dimensions after transformation using ***n\_component*** argument. Here is what I mean – A feature with 5 categories can be represented using N new features similarly, a feature with 100 categories can also be transformed using N new features. Doesn’t this sound amazing?

By default, the Hashing encoder uses **the md5** hashing algorithm but a user can pass any algorithm of his choice. If you want to explore the md5 algorithm, I suggest [this](https://ieeexplore.ieee.org/document/5474379) paper.

import category\_encoders as ce

import pandas as pd

#Create the dataframe

data=pd.DataFrame({'Month':['January','April','March','April','Februay','June','July','June','September']})

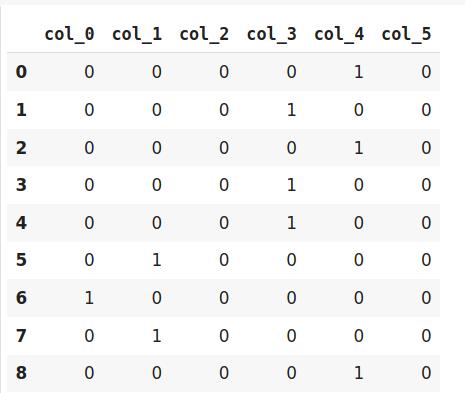
#Create object for hash encoder

encoder=ce.HashingEncoder(cols='Month',n\_components=6)



#Fit and Transform Data

encoder.fit\_transform(data)



Since Hashing transforms the data in lesser dimensions, it may lead to loss of information. Another issue faced by hashing encoder is the **collision.** Since here, a large number of features are depicted into lesser dimensions, hence multiple values can be represented by the same hash value, this is known as a collision.

Moreover, hashing encoders have been very successful in some Kaggle competitions. It is great to try if the dataset has high cardinality features.

## 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

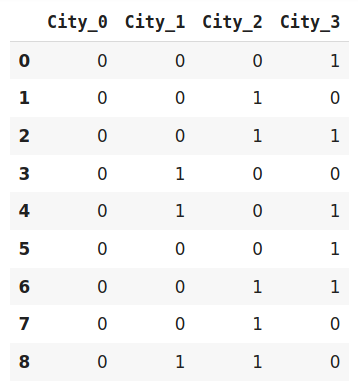
data



#Fit and Transform Data

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

data\_encoded



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.

## Base N Encoding

Before diving into BaseN encoding let’s first try to understand what is Base here?

In the numeral system, the Base or the radix is the number of digits or a combination of digits and letters used to represent the numbers. The most common base we use in our life is 10  or decimal system as here we use 10 unique digits i.e 0 to 9 to represent all the numbers. Another widely used system is binary i.e. the base is 2. It uses 0 and 1 i.e 2 digits to express all the numbers.

For Binary encoding, the Base is 2 which means it converts the numerical values of a category into its respective Binary form. If you want to change the Base of encoding scheme you may use Base N encoder. In the case when categories are more and binary encoding is not able to handle the dimensionality then we can use a larger base such as 4 or 8.

#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 an object for Base N Encoding

encoder= ce.BaseNEncoder(cols=['city'],return\_df=True,base=5)

#Original Data

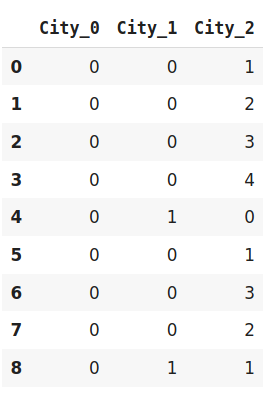
data



#Fit and Transform Data

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

data\_encoded



In the above example, I have used base 5 also known as the Quinary system. It is similar to the example of Binary encoding. While Binary encoding represents the same data by 4 new features the BaseN encoding uses only 3 new variables.

Hence BaseN encoding technique further reduces the number of features required to efficiently represent the data and improving memory usage. The default Base for Base N is 2 which is equivalent to Binary Encoding.

## Target Encoding

Target encoding is a technique used in machine learning and data preprocessing to transform categorical variables into numerical values. Unlike one-hot encoding, which creates binary columns for each category, target encoding calculates and assigns a numerical value to each category based on the relationship between the category and the target variable. Typically used for classification tasks, it replaces the categorical values with their corresponding mean (or other statistical measures) of the target variable within each category.

Target encoding can be effective in capturing valuable information from categorical data while reducing the dimensionality of the feature space, making it suitable for models like decision trees and gradient boosting.

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

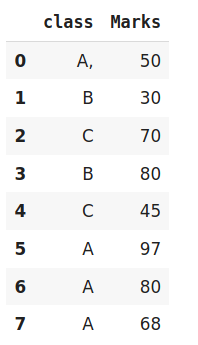
data=pd.DataFrame({'class':['A,','B','C','B','C','A','A','A'],'Marks':[50,30,70,80,45,97,80,68]})

#Create target encoding object

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

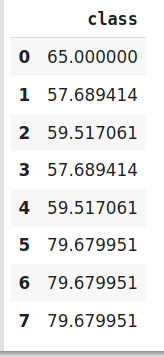
#Original Data

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.
   * In the leave one out encoding, the current target value is reduced from the overall mean of the target to avoid leakage.
   * 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.