Shipra Saxena — Updated On September 24th, 2023

Beginner Machine Learning Python Structured Data Technique

Introduction

The performance of a machine learning model not only depends on the model and the hyperparameters but also on how we process and feed different types of variables to the model. Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.

A typical data scientist spends 70 – 80% of his time cleaning and preparing the data. And converting categorical data is an unavoidable activity. It not only elevates the model quality but also helps in better feature engineering. Now the question is, how do we proceed? Which categorical data encoding method should we use?

In this article, I will be explaining various types of categorical data encoding methods with implementation in Python.

What is Categorical Data?

Since we are going to be working on categorical variables in this article, here is a quick refresher on the same with a couple of examples. Categorical variables are usually represented as 'strings' or 'categories' and are finite in number. Here are a few examples:

1. The city where a person lives: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
2. The department a person works in: Finance, Human resources, IT, Production.
3. The highest degree a person has: High school, Diploma, Bachelors, Masters, PhD.
4. The grades of a student: A+, A, B+, B, B- etc.

In the above examples, the variables only have definite possible values. Further, we can see there are two kinds of categorical data-

---

- **Ordinal Data:** The categories have an inherent order
- **Nominal Data:** The categories do not have an inherent order

In Ordinal data, while encoding, one should retain the information regarding the order in which the category is provided. Like in the above example the highest degree a person possesses, gives vital information about his qualification. The degree is an important feature to decide whether a person is suitable for a post or not.

While encoding Nominal data, we have to consider the presence or absence of a feature. In such a case, no notion of order is present. For example, the city a person lives in. For the data, it is important to retain where a person lives. Here, We do not have any order or sequence. It is equal if a person lives in Delhi or Bangalore.

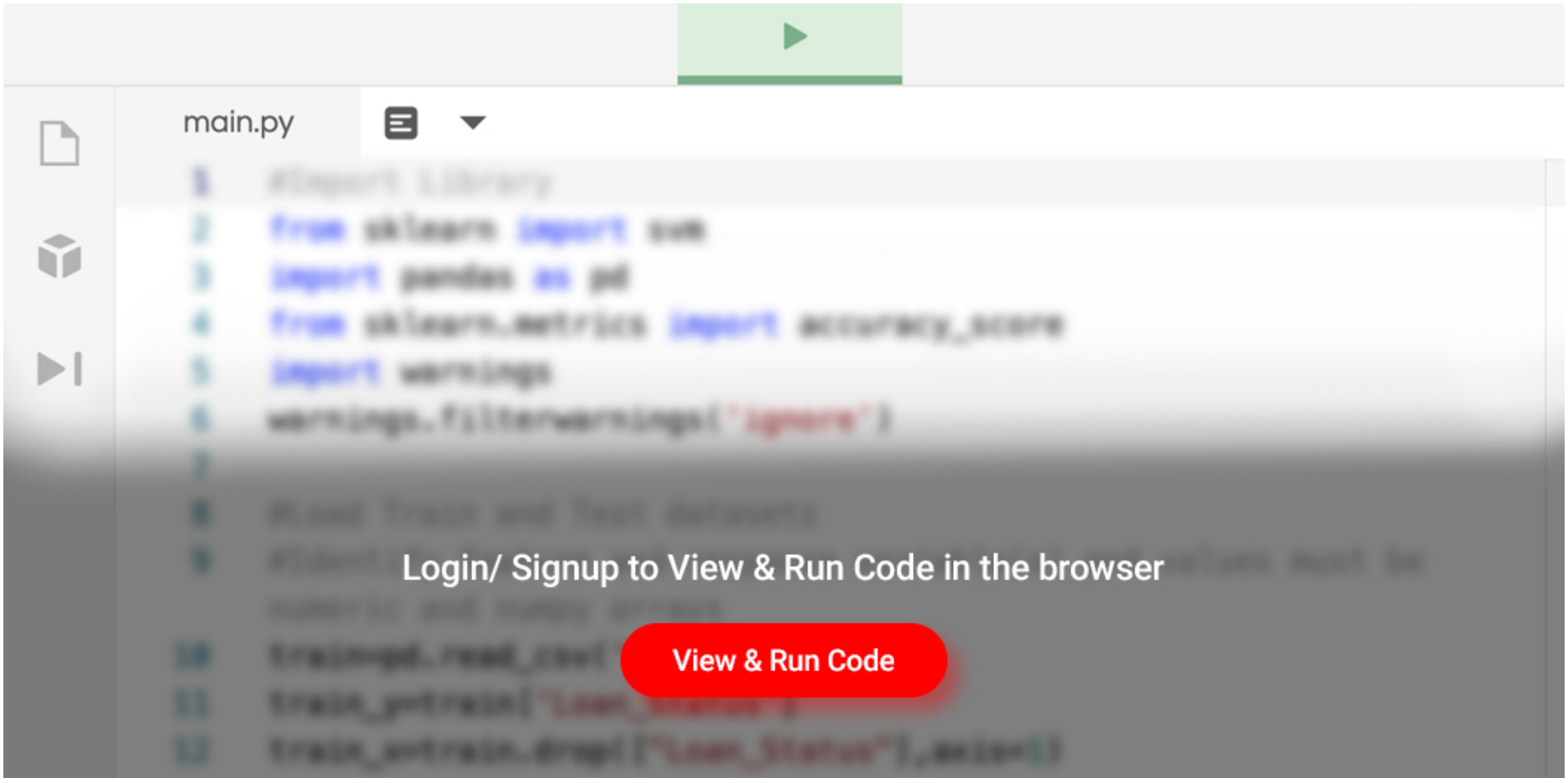
For encoding categorical data, we have a python package category_encoders. The following code helps you install easily.

```
pip install category_encoders
```

is important. Hence encoding should reflect the sequence.

In Label encoding, each label is converted into an integer value. We will create a variable that contains the categories representing the education qualification of a person.

Python Code:



Fit and transform train data

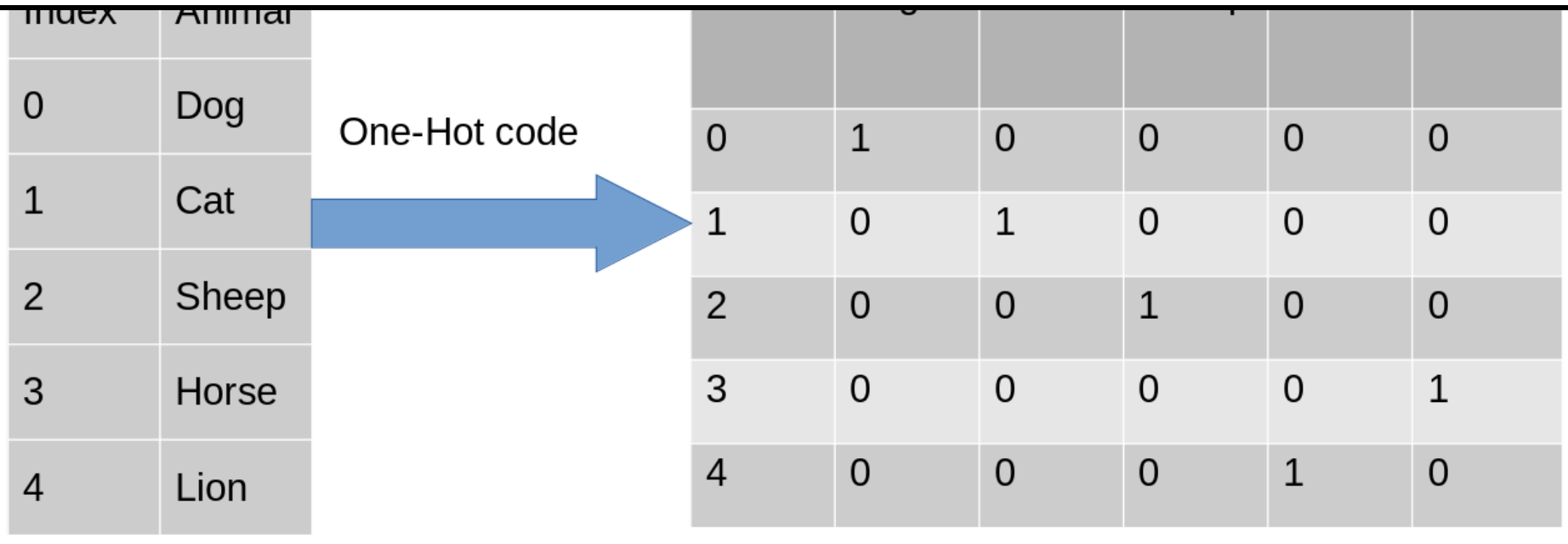
```
df_train_transformed = encoder.fit_transform(train_df)
```

	Degree
0	1
1	4
2	2
3	3
4	3
5	4
6	5
7	1
8	1

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.

These newly created binary features are known as **Dummy variables.** The number of dummy variables depends on the



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, we have 1 in the column of that category and 0 for the others. Let's see how to implement a one-hot encoding in python.

```
import category_encoders as ce
import pandas as pd
data=pd.DataFrame({'City':[
'Delhi','Mumbai','Hydrabad','Chennai','Bangalore','Delhi','Hydrabad','Bangalore','Delhi'
]})

#Create object for one-hot encoding
encoder=ce.OneHotEncoder(cols='City',handle_unknown='return_nan',return_df=True,use_cat_nam
es=True)

#Original Data
data
```

	City
0	Delhi
1	Mumbai
2	Hydrabad
3	Chennai
4	Bangalore
5	Delhi
6	Hydrabad
7	Bangalore
8	Delhi

```
#Fit and transform Data
data_encoded = encoder.fit_transform(data)
data_encoded
```

1	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	1.0
5	1.0	0.0	0.0	0.0	0.0
6	0.0	0.0	1.0	0.0	0.0
7	0.0	0.0	0.0	0.0	1.0
8	1.0	0.0	0.0	0.0	0.0

Now let's move to another very interesting and widely used encoding technique i.e Dummy encoding.

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 (also known as dummy variables). 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.

To understand this better let's see the image below. Here we are coding the same data using both one-hot encoding and dummy encoding techniques. While one-hot uses 3 variables to represent the data whereas dummy encoding uses 2 variables to code 3 categories.

Column	Code
A	100
B	010
C	001

One- Hot Coding

Column	Code
A	10
B	01
C	00

Dummy Code

Let us implement it in python.

```
['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi,'Hyderabad']})

#Original Data
data
```

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad

```
#encode the data
data_encoded=pd.get_dummies(data=data,drop_first=True)
data_encoded
```

	City_Chennai	City_Delhi	City_Hyderabad	City_Mumbai
0	0	1	0	0
1	0	0	0	1
2	0	0	1	0
3	1	0	0	0
4	0	0	0	0
5	0	1	0	0
6	0	0	1	0

Here using *drop_first* argument, we are representing the first label Bangalore using 0.

Drawbacks of One-Hot and Dummy Encoding

One hot encoder and dummy encoder are two powerful and effective encoding schemes. They are also very popular among the data scientists, But may not be as effective when-

1. A large number of levels are present in data. If there are multiple categories in a feature variable in such a case we need a similar number of dummy variables to encode the data. For example, a column with 30 different values will require 30 new variables for coding.
2. If we have multiple categorical features in the dataset similar situation will occur and again we will end to have several binary features each representing the categorical feature and their multiple categories e.g a dataset having 10 or more categorical columns

information.

Also, they might lead to a Dummy variable trap. It is a phenomenon where features are highly correlated. That means using the other variables, we can easily predict the value of a variable.

Due to the massive increase in the dataset, coding slows down the learning of the model along with deteriorating the overall performance that ultimately makes the model computationally expensive. Further, while using tree-based models these encodings are not an optimum choice.

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
```

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad

```
encoder.fit_transform(data)
```

1	1	0.0	1.0	0.0	0.0
2	1	0.0	0.0	1.0	0.0
3	1	0.0	0.0	0.0	1.0
4	1	-1.0	-1.0	-1.0	-1.0
5	1	1.0	0.0	0.0	0.0
6	1	0.0	0.0	1.0	0.0

Effect encoding is an advanced technique. In case you are interested to know more about effect encoding, refer to this 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 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)
```

1	April
2	March
3	April
4	Februay
5	June
6	July
7	June
8	September

```
#Fit and Transform Data
encoder.fit_transform(data)
```

	col_0	col_1	col_2	col_3	col_4	col_5
0	0	0	0	0	1	0
1	0	0	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	1	0	0
5	0	1	0	0	0	0
6	1	0	0	0	0	0
7	0	1	0	0	0	0
8	0	0	0	0	1	0

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.

```
#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
```

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad
7	Mumbai
8	Agra

```
#Fit and Transform Data
data_encoded=encoder.fit_transform(data)
data_encoded
```

	City_0	City_1	City_2	City_3
0	0	0	0	1
1	0	0	1	0
2	0	0	1	1
3	0	1	0	0
4	0	1	0	1
5	0	0	0	1
6	0	0	1	1
7	0	0	1	0
8	0	1	1	0

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?

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
```

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad
7	Mumbai
8	Agra

```
#Fit and Transform Data
data_encoded=encoder.fit_transform(data)
data_encoded
```

1	0	0	2
2	0	0	3
3	0	0	4
4	0	1	0
5	0	0	1
6	0	0	3
7	0	0	2
8	0	1	1

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.

```
#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
```

	class	Marks
0	A,	50
1	B	30
2	C	70
3	B	80
4	C	45
5	A	97
6	A	80
7	A	68

```
#Fit and Transform Train Data
encoder.fit_transform(data['class'],data['Marks'])
```

	class
0	65.000000
1	57.689414
2	59.517061
3	57.689414
4	59.517061
5	79.679951
6	79.679951
7	79.679951

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

mean of the target.

Conclusion

To summarize, encoding categorical data is an unavoidable part of the feature engineering. It is more important to know what coding scheme should we use. Having into consideration the dataset we are working with and the model we are going to use. In this article, we have seen various encoding techniques along with their issues and suitable use cases.

If you want to know more about dealing with categorical variables, please refer to this article-

- *Simple Methods to deal with Categorical Variables in Predictive Modeling*

In case you have any comments please free to reach out to me in the comments below.

Frequently Asked Questions

Q1. Which encoding is best for categorical data?

A. The best encoding method for categorical data depends on the specific dataset and machine learning algorithm. Common encodings include one-hot encoding and label encoding.

Q2. Should I encode categorical variables?

A. Yes, it's essential to encode categorical variables as most machine learning algorithms require numerical data for analysis. Encoding ensures that the model can interpret and learn from categorical features.

Q3. What is the process of encoding categorical data into numerical data called?

A. The process of encoding categorical data into numerical data is called "categorical encoding." It involves transforming categorical variables into a numerical format suitable for machine learning models.

Q4. What is encoding categorical data in Python?

A. In Python, encoding categorical data is typically done using libraries like scikit-learn or pandas. One-hot encoding, label encoding, and target encoding are among the popular techniques used for this purpose.

categorical encoding dummy encoding effect encoding hash encoding label encoding One Hot Encoding target encoding

About the Author

Shipra Saxena

Our Top Authors