


Home > About Python > Learn Python

# Handling Machine Learning Categorical Data with Python Tutorial

Learn the common tricks to handle categorical data and preprocess it to build machine learning models!

Updated Feb 2023 · 28 min read



Moez Ali

Data Scientist, Founder & Creator of PyCaret

TOPICS

- Machine Learning
- Python

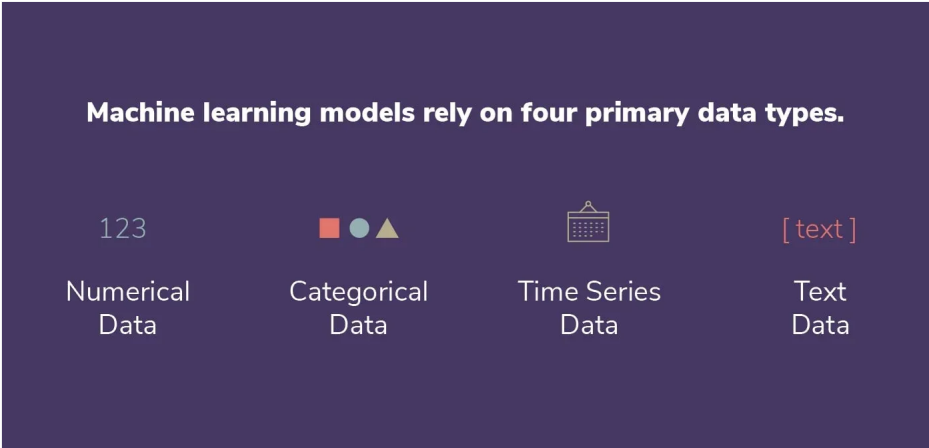


Image Source

## Introduction

Data that can be categorized but lacks an inherent hierarchy or order is known as categorical data. In other words, there is no mathematical connection between the categories. A person's gender (male/female), eye color (blue, green, brown, etc.), type of vehicle they drive (sedan, SUV, truck, etc.), or the kind of fruit they consume (apple, banana, orange, etc.) are examples of categorical data.

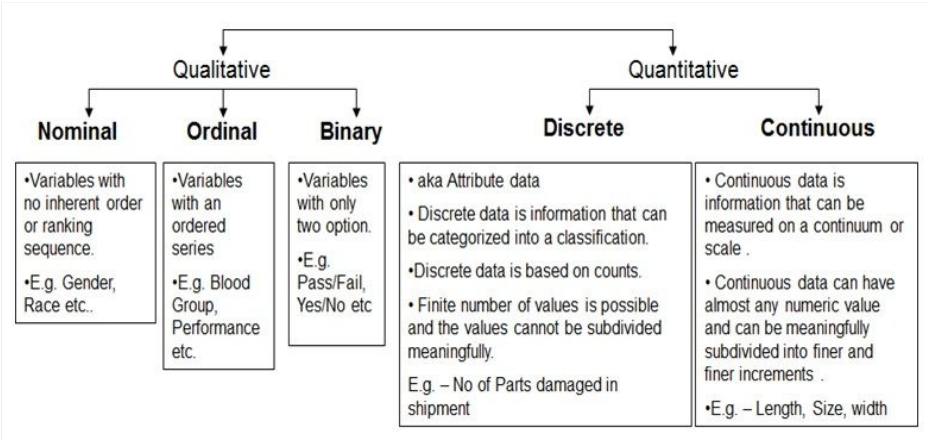
In this tutorial, we'll outline the handling and preprocessing methods for categorical data. Before discussing the significance of preparing categorical data for machine learning models, we'll first define categorical data and its types. Additionally, we'll look at several encoding methods, categorical data analysis and visualization methods in Python, and more advanced ideas like large cardinality categorical data and various encoding methods.

## Categorical Dataset

Information is represented using two different forms of data: categorical data and numeric data. Data that may be categorized or grouped together is referred to as categorical data. Men and women fall into the gender category, the colors red, green, and blue fall into the colors category, and the countries category might include the US, Canada, Mexico, etc.

**Numeric data** refers to data that can be expressed as a number. Examples of numeric data include height, weight, and temperature.

In simple terms, categorical data is information that can be put into categories, while numeric data is information that can be expressed as a number. Since the majority of machine learning algorithms are created to operate with numerical data, categorical data is handled differently from numerical data in this field. Before categorical data can be utilized as input to a machine learning model, it must first be transformed into numerical data. This process of converting categorical data into numeric representation is known as **encoding**.



Qualitative and Quantitative Data - Image Source

There are two types of categorical data: nominal and ordinal.

### Nominal data

Nominal data is categorical data that may be divided into groups, but these groups lack any intrinsic hierarchy or order. Examples of nominal data include brand names (Coca-Cola,



Pepsi, Sprite), varieties of pizza toppings(pepperoni, mushrooms, onions), and hair color (blonde, brown, black, etc.).

Ordinal data

Ordinal data, on the other hand, describes information that can be categorized and has a distinct order or ranking. Levels of education (high school, bachelor's, master's), levels of work satisfaction (extremely satisfied, satisfied, neutral, unsatisfied, very unsatisfied), and star ratings (1-star, 2-star, 3-star, 4-star, 5-star) are a few examples of ordinal data.

By giving each category a numerical value that reflects its order or ranking, ordinal data can be transformed into numerical data and used in machine learning. For algorithms that are sensitive to the size of the input data, this may be helpful.

The distinction between nominal and ordinal data isn't always obvious in practice and may vary depending on the particular use case. For instance, while some people may view "star rating" as ordinal data, others may view it as nominal data. The most important thing is to be aware of the nature of your data and select the encoding strategy that best captures the relationships in your data.

Understanding data types in pandas

The widely used Python open-source library pandas is used for data analysis and manipulation. It has strong capabilities for dealing with structured data, including as data frames and series that can deal with tabular data with labeled rows and columns.

pandas also provides several functions to read and write different file types (csv, parquet, database, etc.). When you read a file using pandas, each column is assigned a data type based on the inference. Here are all the data types pandas can possibly assign:

- 1. **Numeric:** This includes integers and floating-point numbers. Numeric data is typically used for quantitative analysis and mathematical operations.
- 2. **String:** This data type is used to represent textual data such as names, addresses, and descriptions.
- 3. **Boolean:** This data type can only have two possible values: True or False. Boolean data is often used for logical operations and filtering.
- 4. **Datetime:** This data type is used to represent dates and times. pandas has powerful tools for manipulating datetime data.
- 5. **Categorical:** This data type represents data that takes on a limited number of values. Categorical data is often used for grouping and aggregating data.
- 6. **Object:** This data type is a catch-all for data that does not fit into the other categories. It can include a variety of different data types, such as lists, dictionaries, and other objects.

Pandas dtype	Python type	NumPy type	Usage
object	str	string_, unicode_	Text
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

Image Source

Data that doesn't fit into the other data types, including strings, mixed kinds, or other objects, is represented by the **category** and **object** data types in pandas. However, they have a few significant distinctions that have an impact on how well they work and perform.

Categorical Data Type

The categorical data type was created for information that only has a few possible values, like categories or labels. Internally, categorical data is represented as a collection of numbers, which can speed up some operations and conserve memory in comparison to the corresponding object data type. Additionally, categorical data can be arranged logically and facilitates effective grouping and aggregating procedures.

Object Data Type

On the other hand, the object data type serves as a catch-all for information that does not fit into the other data types. Lists, dictionaries, and other objects are just a few examples of the many various types of data that it may include. Although object data has a great deal of flexibility, it can also be slower and consume more memory than categorical data of the same size and can't be subjected to some of the specialized operations that are possible with category data.

In general, you might want to think about utilizing the categorical data type if your data has a small number of possible values and you intend to conduct a lot of grouping or aggregating. The object data type is typically a secure option in all other cases. The ideal data type, however, ultimately depends on your unique use case and the properties of your data.

Let's see an example by reading a [csv file from GitHub](#).

```
# read csv using pandas
data = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/data

# check the head of dataframe
data.head()
```

POWERED BY DATACAMP WORKSPACE

Output:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171

Can you identify which of these columns are categorical vs. numeric? Well, all the columns in this example are categorical except for `Carat Weight` and `Price.` Let's see if we are right about this by checking the default data types.

```
# check the data types
data.dtypes
```



🔗 Explain code

OpenAI

Output:

```
Carat Weight    float64
Cut              object
Color           object
Clarity         object
Polish          object
Symmetry        object
Report          object
Price           int64
dtype: object
```

Notice how `Price` is assigned `int64` type, `Carat Weight` as `float64`, and the rest of the columns are objects, exactly as we expected.

## Analyzing Categorical Features in Python

There are a few functions in pandas, a popular data analysis library in Python, that allow you to quickly analyze categorical data types in your dataset. Let us examine them one by one:

### Value Counts

`value\_counts()` is a function in the pandas library that returns the frequency of each unique value in a categorical data column. This function is useful when you want to get a quick understanding of the distribution of a categorical variable, such as the most common categories and their frequency.

```
# read csv using pandas
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/data')

# check value counts of Cut column
data['Cut'].value_counts()
```



🔗 Explain code

OpenAI

Output:

```

Ideal          2482
Very Good      2428
Good           708
Signature-Ideal 253
Fair           129
Name: Cut, dtype: int64
```

If you want to visualize distribution you can use `plotly` library to draw an interactive bar plot.

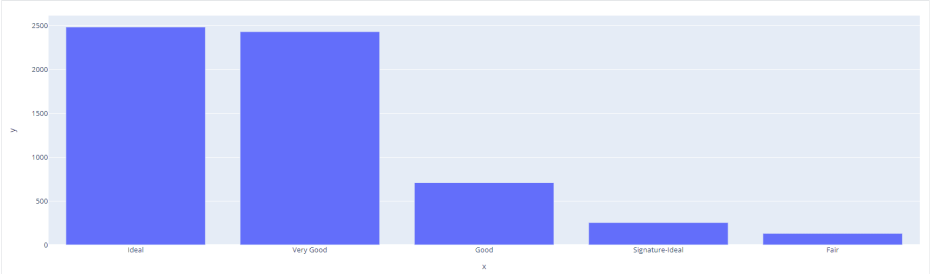
```
import plotly.express as px
cut_counts = data['Cut'].value_counts()
fig = px.bar(x=cut_counts.index, y=cut_counts.values)
fig.show()
```



🔗 Explain code

OpenAI

Output:



### Group by

`groupby()` is a function in Pandas that allows you to group data by one or more columns and apply aggregate functions such as sum, mean, and count. This function is useful when you want to perform more complex analysis on categorical data, such as computing the average of a numeric variable for each category. Let's see an example:

```
# read csv using pandas
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/data')

# average carat weight and price by Cut
data.groupby(by = 'Cut').mean()
```

🔮 Explain code



Output:

	Carat Weight	Price
Cut		
Fair	1.058682	5886.178295
Good	1.268927	9326.656780
Ideal	1.382293	13127.331185
Signature-Ideal	1.205217	11541.525692
Very Good	1.332941	11484.696870

It will only return a data frame with numeric columns only. The `by` parameter inside the `groupby` method defines the column on which you want to perform group by operation, and then `mean()` outside of parentheses is the aggregation method.

The output can be interpreted as the average price of a diamond with fair cut is 5,886, and the average weight is 1.05 compared to the average price of 11,485 for a diamond with a Very Good cut.

Cross tab

`crosstab()` is a function in pandas that creates a cross-tabulation table, which shows the frequency distribution of two or more categorical variables. This function is useful when you want to see the relationship between two or more categorical variables, such as how the frequency of one variable is related to another variable.

```
# read csv using pandas
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/data')

# cross tab of Cut and Color
pd.crosstab(index=data['Cut'], columns=data['Color'])
```

🔮 Explain code



Output:

	Color	D	E	F	G	H	I
Cut							
Fair	12	32	24	21	24	16	
Good	74	110	133	148	128	115	
Ideal	280	278	363	690	458	413	
Signature-Ideal	30	35	38	64	45	41	
Very Good	265	323	455	578	424	383	

The output from the crosstab function in pandas is a table that shows the frequency distribution of two or more categorical variables. Each row of the table represents a unique category in one of the variables, and each column represents a unique category in the other variable. The entries in the table are the frequency counts of the combinations of categories in the two variables.

Pivot Table

`pivot\_table()` is a function in Pandas that creates pivot tables, which are similar to cross-tabulation tables but with more flexibility. This function is useful when you want to analyze multiple categorical variables and their relationship to one or more numeric variables. Pivot tables allow you to aggregate data in multiple ways and display the results in a compact form.

```
# read csv using pandas
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/data')

# create pivot table
pd.pivot_table(data, values='Price', index='Cut', columns='Color', aggfunc=np.me
```

🔮 Explain code



Output:

	Color	D	E	F	G	H	I
Cut							
Fair	6058.250000	5370.625000	6063.625000	7345.523810	5908.500000	4573.187500	
Good	10058.716216	8969.545455	9274.007519	9988.614865	9535.132812	8174.113043	
Ideal	18461.953571	12647.107914	14729.426997	13570.310145	11527.700873	9459.588378	
Signature-Ideal	19823.100000	11261.914286	13247.947368	10248.296875	9112.688889	8823.463415	
Very Good	13218.826415	12101.910217	12413.905495	12354.013841	10056.106132	8930.031332	

This table shows the average price of each diamond cut for each color. The rows represent the different diamond cut, the columns represent the different diamond colors, and the entries in the table are the average price of the diamond.

The `pivot_table` function is useful when you want to summarize and compare the numerical data across multiple variables in a table format. The function allows you to aggregate the data using various functions (such as mean, sum, count, etc.) and organize it into a format that is easy to read and analyze.

## Encoding Categorical Features in Python

Categorical data cannot typically be directly handled by machine learning algorithms, as most algorithms are primarily designed to operate with numerical data only. Therefore, before categorical features can be used as inputs to machine learning algorithms, they must be encoded as numerical values.

There are several techniques for encoding categorical features, including one-hot encoding, ordinal encoding, and target encoding. The choice of encoding technique depends on the specific characteristics of the data and the requirements of the machine learning algorithm being used.

### One-hot encoding

One hot encoding is a process of representing categorical data as a set of binary values, where each category is mapped to a unique binary value. In this representation, only one bit is set to 1, and the rest are set to 0, hence the name "one hot." This is commonly used in machine learning to convert categorical data into a format that algorithms can process.

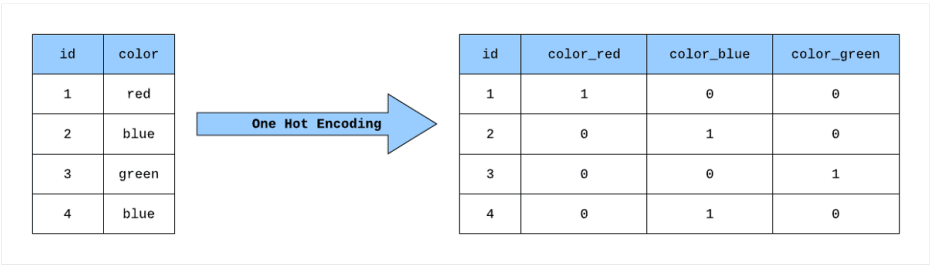


Image Source

### pandas categorical to numeric

One way to achieve this in pandas is by using the `pd.get_dummies()` method. It is a function in the Pandas library that can be used to perform one-hot encoding on categorical variables in a DataFrame. It takes a DataFrame and returns a new DataFrame with binary columns for each category. Here's an example of how to use it:

Suppose we have a data frame with a column "fruit" containing categorical data:

```
import pandas as pd

# generate df with 1 col and 4 rows
data = {
    "fruit": ["apple", "banana", "orange", "apple"]
}

# show head
df = pd.DataFrame(data)
df.head()
```

🔗 Explain code

🔗 OpenAI

Output:

	fruit
0	apple
1	banana
2	orange
3	apple

```
# apply get_dummies function
df_encoded = pd.get_dummies(df["fruit"])
df_encoded.head()
```

🔗 Explain code

🔗 OpenAI

Output:

	apple	banana	orange
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0

Even though `pandas.get_dummies` is straightforward to use, a more common approach is to use `OneHotEncoder` from the `sklearn` library, especially when you are doing machine learning tasks. The primary difference is `pandas.get_dummies` cannot learn encodings; it can only perform one-hot-encoding on the dataset you pass as an input. On the other hand, `sklearn.OneHotEncoder` is a class that can be saved and used to transform other incoming datasets in the future.

```
import pandas as pd

# generate df with 1 col and 4 rows
data = {
    "fruit": ["apple", "banana", "orange", "apple"]
}
```

```
}

# one-hot-encode using sklearn
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
encoded_results = encoder.fit_transform(df).toarray()
```

✦ Explain code

OpenAI

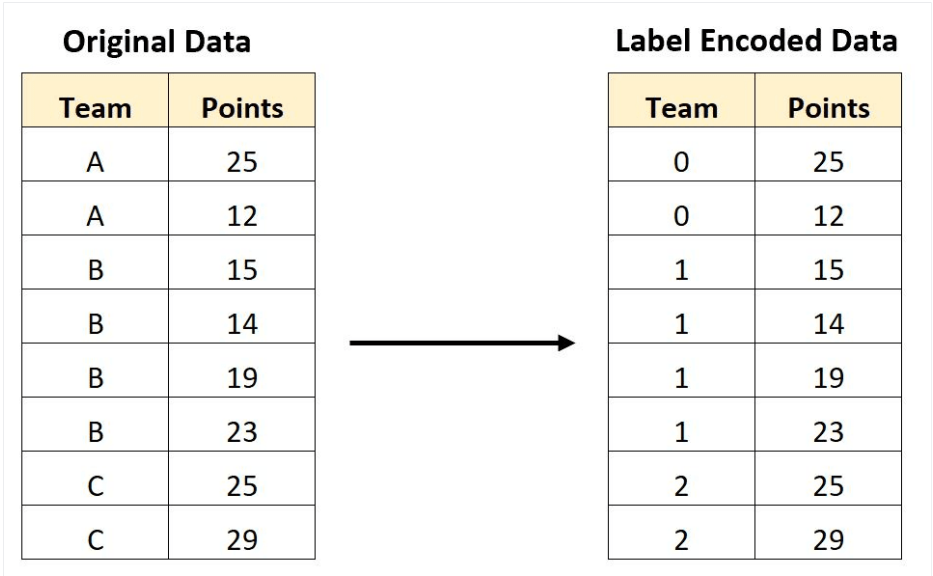
Output:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

Label encoding

Label encoding is a technique for encoding categorical variables as numeric values, with each category assigned a unique integer. For example, suppose we have a categorical variable "color" with three categories: "red," "green," and "blue." We can encode these categories using label encoding as follows (red: 0, green: 1, blue: 2).

Label encoding can be useful for some machine learning algorithms that require numeric inputs, as it allows categorical data to be represented in a way that the algorithms can understand. However, it's important to keep in mind that label encoding introduces an arbitrary ordering of the categories, which may not necessarily reflect any meaningful relationship between them. In some cases, this can lead to problems in the analysis, especially if the ordering is interpreted as having some kind of ordinal relationship.



Label Encoded Data. [Image Source](#)

Comparison of One-hot and Label Encoding

One-hot encoding and label encoding are both techniques for encoding categorical variables as numeric values, but they have different properties and are appropriate for different use cases.

One-hot encoding represents each category as a binary column, with a 1 indicating the presence of the category and a 0 indicating its absence. For example, suppose we have a categorical variable "color" with three categories: "red," "green," and "blue." One-hot encoding would represent this variable as three binary columns:

color	red	green	blue
red	1	0	0
green	0	1	0
blue	0	0	1

One-hot encoding is appropriate when the categories do not have an intrinsic ordering or relationship with each other. This is because one-hot encoding treats each category as a separate entity with no relation to the other categories. One-hot encoding is also useful when the number of categories is relatively small, as the number of columns can become unwieldy for very large numbers of categories.

Label encoding, on the other hand, represents each category as a unique integer. For example, the "color" variable with three categories could be label-encoded as:

color	color_code
red	0
green	1
blue	2

Label encoding is appropriate when the categories have a natural ordering or relationship with each other, such as in the case of ordinal variables like "small," "medium," and "large." In these cases, the integer values assigned to the categories should reflect the ordering of the categories. Label encoding can also be useful when the number of categories is very large, as it reduces the dimensionality of the data.

In general, one-hot encoding is more commonly used in machine learning applications, as it is more flexible and avoids the problems of ambiguity and arbitrary ordering that can arise with label encoding. However, label encoding can be useful in certain contexts where the categories have a natural ordering or when dealing with very large numbers of categories.

Dealing with High Cardinality Categorical Data

High cardinality refers to a large number of unique categories in a categorical feature. Dealing with high cardinality is a common challenge in encoding categorical data for machine learning models. High cardinality can lead to sparse data representation and can



have a negative impact on the performance of some machine learning models. Here are some techniques that can be used to deal with high cardinality in categorical features:

Combining Rare Categories

This involves combining infrequent categories into a single category. This reduces the number of unique categories and also reduces the sparsity in the data representation.

Target Encoding

Target encoding replaces the categorical values with the mean target value of that category. It provides a more continuous representation of the categorical data and can help capture the relationship between the categorical feature and the target variable.



Image Source

Weight of Evidence (WOE) Encoding

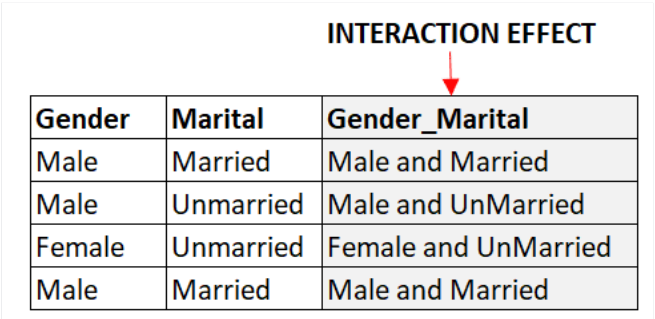
Weight of Evidence encoding is similar to target encoding, but it takes into account the distribution of the target variable for each category. The WOE of a category is calculated as the logarithm of the ratio of the target's mean for the category over the mean for the entire population.

Feature Engineering for Categorical Data

Feature engineering is an important step in preparing data for machine learning models. It involves creating new features from the existing features to improve the performance of the models. Here are a few ways you can perform feature engineering on categorical data:

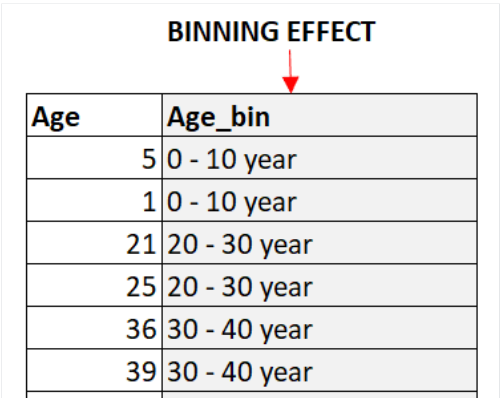
Creating interaction variables

Interaction variables are new features created by combining two or more existing features. For example, if we have two categorical features, 'Gender' and 'Marital Status,' we can create a new feature, 'Gender-Marital Status,' to capture the interaction between the two features. This can help to capture non-linear relationships between the features and the target variable.



Binning numerical variables

Binning is the process of dividing continuous numerical variables into discrete bins. This can help to reduce the number of unique values in the feature, which can be beneficial for encoding categorical data. Binning can also help to capture non-linear relationships between the features and the target variable.



Encoding cyclical variables

Cyclical variables are variables that repeat over a specific period. For example, the hour of the day is a cyclical variable as it repeats every 24 hours. Encoding cyclical variables can help to capture the periodic patterns in the data. One common approach to encoding cyclical variables is to create two new features, one representing the sine of the variable and the other representing the cosine of the variable.

Conclusion

Handling categorical data is an important aspect of many machine learning projects. In this tutorial, we have explored various techniques for analyzing and encoding categorical variables in Python, including one-hot encoding and label encoding, which are two commonly used techniques.

We began by introducing the concept of categorical data and why it is important to handle it properly in machine learning models. We then provided a step-by-step guide on how to perform one-hot encoding using both pandas and scikit-learn, along with code examples to illustrate the process.

This tutorial also discusses some advanced concepts like dealing with high cardinality categorical data, feature engineering, WOE encoding, and more. If you would like to deep dive further into this topic, check out our course, [Working with Categorical Data in Python](#).

If you prefer R language, you may be interested in our courses, [Categorical Data in the Tidyverse](#) or [Inference for Categorical Data in R](#). Both of these are amazing courses for R users.

Master your data skills with DataCamp

Learn the skills you need at your own pace—from non-coding essentials to data science and machine learning.

By continuing, you accept our [Terms of Use](#), our [Privacy Policy](#) and that your data is stored in the USA.

Email Address

Email Address

Password

Password



Start Learning

TOPICS

Machine Learning Python

Learn more about Python

Cleaning Data in Python

Beginner 4 hr 92.3K

Learn to diagnose and treat dirty data and develop the skills needed to transform your raw data into accurate insights!

See Details →

Start Course

See More →

Related



Machine Learning Engineer Salaries in 2023

Natassha Selvaraj



What is Continuous Learning? Revolutionizing Machine...

Yolanda Ferreira



What is Natural Language Processing (NLP)? A...

Matt Crabtree

See More →

Grow your data skills with DataCamp for Mobile

Make progress on the go with our mobile courses and daily 5-minute coding challenges.



LEARN

Learn Python

Learn R

Learn AI

Learn SQL

Learn Power BI

Learn Tableau

Learn Data Engineering

Assessments



- Career Tracks
- Skill Tracks
- Courses
- Data Science Roadmap

DATA COURSES

- Upcoming Courses
- Python Courses
- R Courses
- SQL Courses
- Power BI Courses
- Tableau Courses
- Spreadsheets Courses
- Data Analysis Courses
- Data Visualization Courses
- Machine Learning Courses
- Data Engineering Courses

WORKSPACE

- Get Started
- Templates
- Integrations
- Documentation

CERTIFICATION

- Certifications
- Data Scientist
- Data Analyst
- Data Engineer
- Hire Data Professionals

RESOURCES

- Resource Center
- Upcoming Events
- Blog
- Code Alongs
- Tutorials
- Open Source
- RDocumentation
- Course Editor
- Book a Demo with DataCamp for Business
- Data Portfolio
- Portfolio Leaderboard

PLANS

- Pricing
- For Business
- For Universities
- Discounts, Promos & Sales
- DataCamp Donates

SUPPORT

- Help Center
- Become an Instructor
- Become an Affiliate

ABOUT

- About Us
- Learner Stories
- Careers
- Press

[Leadership](#)  
[Contact Us](#)



[Privacy Policy](#)   [Cookie Notice](#)   [Do Not Sell My Personal Information](#)   [Accessibility](#)   [Security](#)   [Terms of Use](#)

© 2023 DataCamp, Inc. All Rights Reserved.