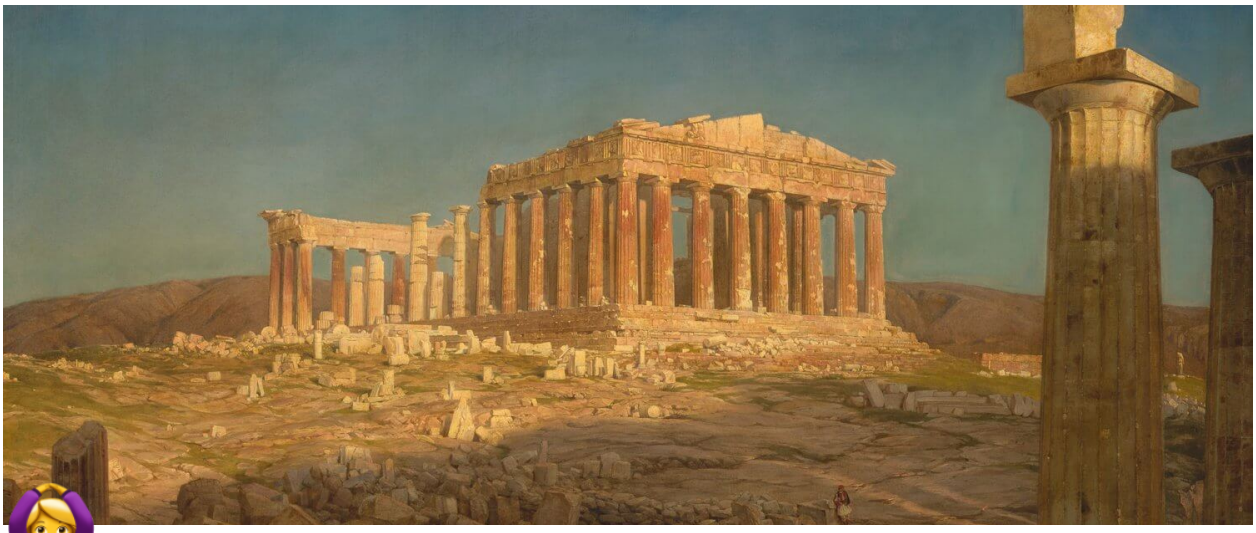
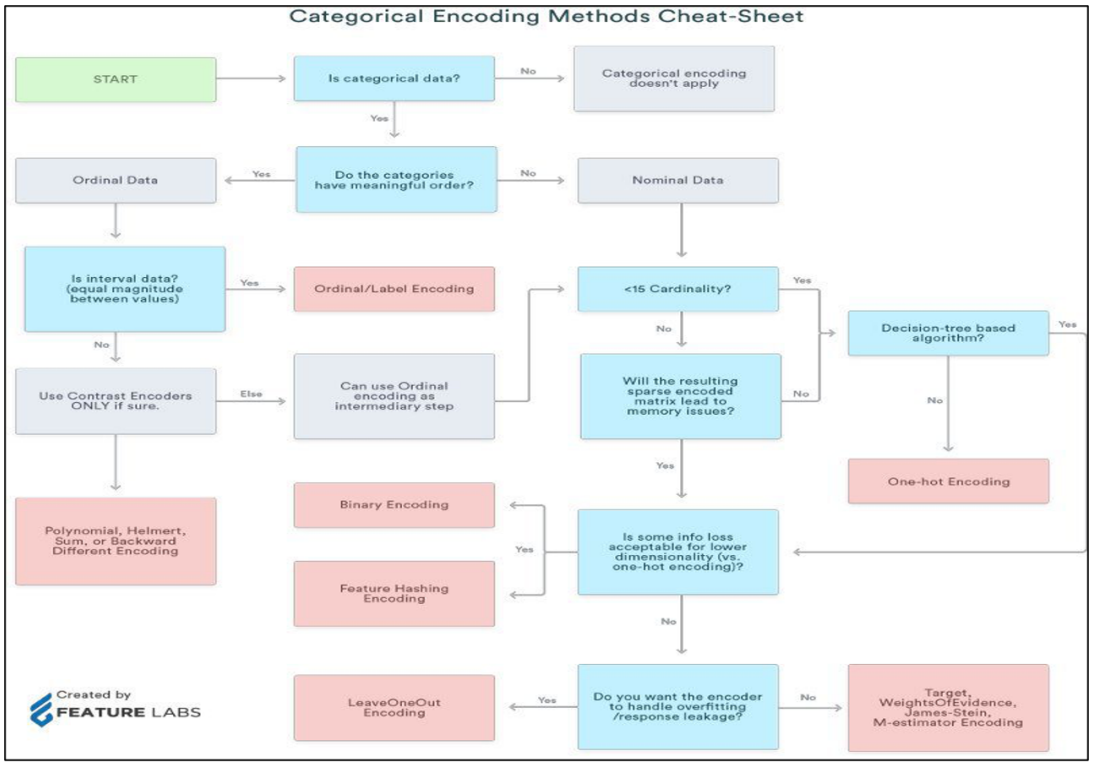


Handling Categorical Datatype (Encoding Methods)

Created by : Eng. Amjaad Altakhaineh

🗺️ Let's dive into this map first:

↗



Introduction:

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 Main focus of this Notebook is to understand

- **What is Categorical Data and Why to Encode Data**
- **Different Data Encoding Techniques**
- **How to Implement it**.

What is Categorical Data?

When we collect data, we often encounter different types of variables. One such type is categorical variables. Categorical variables are usually represented as 'strings' or 'categories' and are finite in number.

There are two types of categorical data -

- ***Ordinal Data***
- ***Nominal Data***

Here are a few examples of categorical variables:

- **Places**: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
- **Departments**: Finance, Human resources, IT, Production.
- **Grades**: A, A-, B+, B, B- etc.

Ordinal Data:

The categories of ordinal data have an **Inherent Order**. This means that the categories can be **Ranked** or ordered from highest to lowest or vice versa.

For example, the variable "highest degree a person has" is an ordinal variable. The categories (High school, Diploma, Bachelors, Masters, PhD) can be ranked in order of the level of education attained.

Nominal Data:

The categories of nominal data **do not have an Inherent Order**. This means that the categories cannot be ranked or ordered.

For example, the variable "city where a person lives" is a nominal variable. The categories (Delhi, Mumbai, Ahmedabad, Bangalore, etc.) cannot be ranked or ordered.

What is Data Encoding?

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.

Why it is Important?

- Most machine learning algorithms work only with numerical data, so categorical variables (such as text labels) must be transformed into numerical values.
- This allows the model to identify patterns in the data and make predictions based on those patterns.
- Encoding also helps to prevent bias in the model by ensuring that all features are equally weighted.
- 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

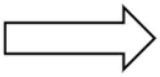
Let's take a closer look at each of these methods.

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

Places
New York
Boston
Chicago
California
New Jersey



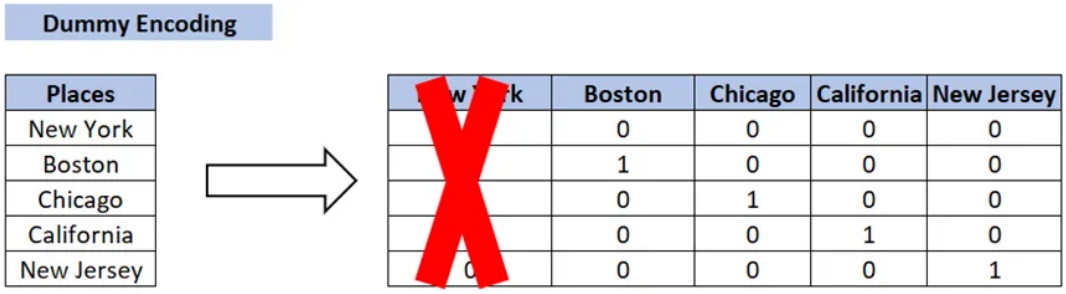
New York	Boston	Chicago	California	New Jersey
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

```
# 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.
-



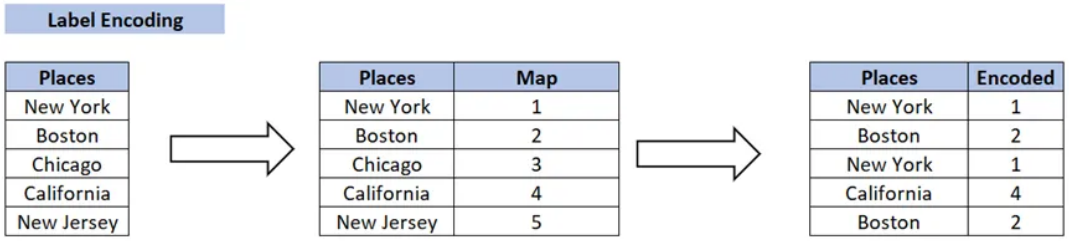
> One-Hot Encoding vs Dummy Encoding:
>
> **One-Hot Encoding** — N categories in a variable, **N** binary variables.
>
> **Dummy encoding** — N categories in a variable, **N-1** binary variables.

```
# 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:

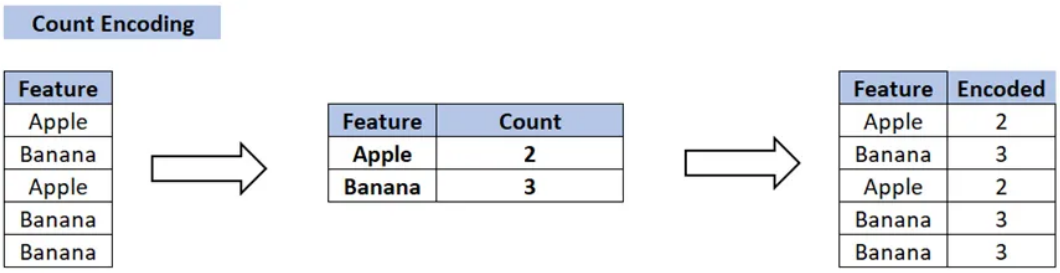
- Each unique category is assigned a **Unique Integer** value.
- This is a simpler encoding method, but it has a **Drawback** in that the assigned integers may be **misinterpreted** by the machine learning algorithm **as having an Ordered Relationship** when in fact they **do not**.
-




```
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'])
```

Count Encoding:

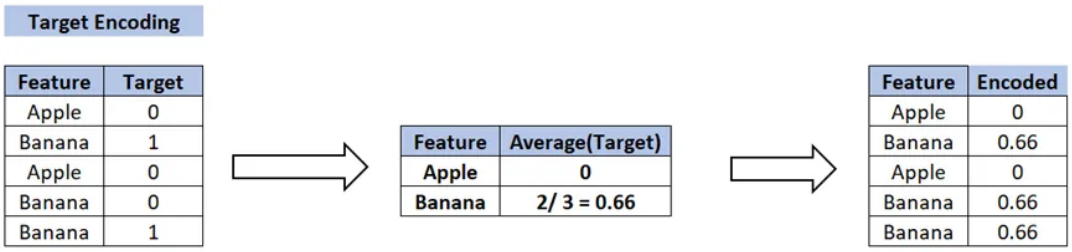
- Count Encoding is a method for encoding categorical variables by **counting the number of times a category appears** in the dataset.
- For example, if a variable has categories ‘A’, ‘B’ and ‘C’ and category ‘A’ appears 10 times in the dataset, it will be assigned a value of 10.




```
# 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.
-




```
# Create a sample dataframe with categorical data and target
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green'],
                   'target': [0, 1, 0, 1, 0]})
print(f"Before Encoding the Data:\n\n{df}\n")
```

```
# Calculate the mean target value for each category
target_mean = df.groupby('color')['target'].mean()

# Replace the categorical data with the mean target value
df['color_label'] = df['color'].map(target_mean)


print(f"After Encoding the Data:\n\n{df}")
```

Conclusion:

Data Encoding is an important step in the pre-processing of data for machine learning algorithms. The choice of encoding method depends on the type of data and the problem being solved. One-Hot Encoding is the most commonly used method, but other methods like Ordinal Encoding, Binary Encoding, and Count Encoding may also be used in certain situations.

Let's examine the following table which provides a comprehensive comparison of different encoding techniques:

Technique	Definition	Limitations	Required Libraries/Tools	Use Case	Reference
Label Encoding	In label encoding, each category is assigned a value from 1 through N where N is the number of categories for the feature. There is no relation or order between these assignments.	The Country names do not have an order or rank. However, label encoding can create an artificial order that the model may misinterpret as meaningful. This can lead to inaccuracies, e.g., India < Japan < US.	from sklearn.preprocessing import LabelEncoder	1. The categorical feature is ordinal (e.g., Jr. kg, Sr. kg, Primary school). 2. Handle large numbers of categories efficiently compared to one-hot encoding.	(analyticsvidhya, 2022) (maxhalford, 2022)
One-Hot Encoding	Each level of a categorical feature is mapped to a binary variable (dummy variable), where 0	Dummy Variable Trap: The outcome of one variable can easily be predicted using others. Can lead to memory	from sklearn.preprocessing import OneHotEncoder onehotencoder = OneHotEncoder()	1. The categorical feature is not ordinal (e.g., country names). 2. Works best when the	(analyticsvidhya, 2022) (maxhalford, 2022)

Technique	Definition	Limitations	Required Libraries/Tools	Use Case	Reference
	indicates absence and 1 indicates presence.	inefficiency when the number of categories is high.		number of categorical features is low.	
Dummy Encoding	Similar to one-hot encoding but uses N-1 features to represent N categories.	Creates many variables if the categorical feature has multiple categories, leading to inefficiency. Handling unknown categories in test data requires additional effort to ensure consistency with training data.	pd.get_dummies(data=data, drop_first=True)	Useful for datasets where efficient memory usage is important, and categories are consistent between training and test sets.	(pythonsimplified, 2022)
Target Encoding	Replaces categories with their corresponding probability (if target is categorical) or average (if target is numerical).	Prone to overfitting, especially when the number of values used in averages is low. Can cause data leakage if applied incorrectly.	from category_encoders import TargetEncoder targetencoder = TargetEncoder() pip install category_encoders	1. High-cardinality features: Useful for features with many categories. 2. Domain-specific features: Helps identify the relationship between features and the target variable based on domain knowledge.	(saedsayad, 2022) (kaggle, 2022)