Experiment – 5

Encoding Categorical Variables: Convert categorical variables into numerical representations using techniques like one-hot encoding, label encoding, or target encoding.

Date Set: *We will use the data in a csv file, namely* ***data.csv****, which is the data set on which we will implement procedure to handle missing data. This data set belongs to a retail company that collected some data from their customers, whether or not they purchased a certain product. Here, each of the rows correspond to different customers. For each of these customers the company gathered their country, their age, salary, and whether or not they purchased their product.*

Objective:

Convert categorical variables into numerical representations.

Tasks:

1. Explore different encoding techniques (one-hot encoding/ label encoding/ target encoding).
2. Apply each encoding technique to categorical variables in the dataset.
3. Compare the impact of encoding techniques on model performance.

Encoding categorical variables refers to the process of converting categorical data, which represents categories or labels, into a numerical format that can be used for machine learning algorithms. Categorical variables are typically non-numeric and can include data such as gender, colour, or country. There are several techniques for encoding categorical variables:

1. **One-Hot Encoding:** This technique creates binary columns for each category in the categorical variable. Each column represents a category, and a value of 1 indicates the presence of that category while 0 indicates absence. One-hot encoding is suitable when there is no inherent order or ranking among the categories.
2. **Label Encoding:** Label encoding assigns a unique numerical value to each category in the variable. It essentially replaces each category with a numerical label. This technique is suitable for ordinal categorical variables where there is an inherent order or ranking among the categories.
3. **Ordinal Encoding:** Similar to label encoding, ordinal encoding assigns numerical values to categories based on their order or rank. However, in ordinal encoding, the numerical values are typically assigned based on predefined rules or mapping dictionaries.
4. **Target Encoding:** Target encoding replaces each category with the mean of the target variable for that category. It is often used in classification tasks and can help capture the relationship between categorical variables and the target variable.

The choice of encoding technique depends on the nature of the categorical variable, the machine learning algorithm being used, and the specific requirements of the task at hand.

Python Program:

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv(r'C:\Users\...\GHRCEM\Dataset\Data.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

```
print(dataset.values)
# prints the entire dataset
```

Output:

```
[['France' 44.0 72000.0 'No']
 ['Spain' 27.0 48000.0 'Yes']
 ['Germany' 30.0 54000.0 'No']
 ['Spain' 38.0 61000.0 'No']
 ['Germany' 40.0 nan 'Yes']
 ['France' 35.0 58000.0 'Yes']
 ['Spain' nan 52000.0 'No']
 ['France' 48.0 79000.0 'Yes']
 ['Germany' 50.0 83000.0 'No']
 ['France' 37.0 67000.0 'Yes']]
```

```
print(X)
```

Output:

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
print(Y)
```

Output:

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Taking care of missing data

```
from sklearn.impute import SimpleImputer as si
imputer = si(missing_values = np.nan, strategy = 'mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
print(X)
```

Output:

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Encoding Categorical Data

Encoding the Independent Variable

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

Note:

- **ColumnTransformer**: This class allows you to selectively apply transformers (such as scaling, one-hot encoding, etc.) to different columns or subsets of columns in your dataset.
- **OneHotEncoder**: This transformer is used for converting categorical variables into a form that can be provided to ML algorithms to do a better job in prediction.

```
ct = ColumnTransformer(transformers = [('encoder',
OneHotEncoder(), [0])], remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
```

Note:

- Here, a **ColumnTransformer** object **ct** is created. While creating **ct** object, **ColumnTransformer** class takes 2 parameters: **transformers** and **remainder.**
- **transformers** parameter: It's a list of tuples. Each tuple contains three elements:
    - Name: A name to identify the transformation (e.g., '**encoder**').
    - Transformer: The transformer to be applied. In this case, it's **OneHotEncoder()**.
    - Columns: The indices of the columns to which the transformation should be applied. **[0]** indicates the first column in the dataset (column indexing starts from 0).
- **remainder** parameter: It specifies what to do with the columns not explicitly transformed. '**passthrough**' means that those columns will be left untouched.

```
print(X)
```

Output:

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
```

```
[1.0 0.0 0.0 48.0 79000.0]
[0.0 1.0 0.0 50.0 83000.0]
[1.0 0.0 0.0 37.0 67000.0]]
```

Encoding the Dependent Variable

```
from sklearn.preprocessing import LabelEncoder
```

Note:
This line imports the **LabelEncoder** class from the **sklearn.preprocessing** module. This class is used for converting categorical labels into numerical labels. It assigns a unique integer to each category.

```
le = LabelEncoder()
Y = le.fit_transform(Y)
```

Note:

- Here, a **LabelEncoder** object **le** is created. This object will be used to perform the label encoding. **LabelEncoder** is used when dealing with categorical target variables (labels) in a supervised learning task. It converts categorical labels into numerical format, which is essential for many machine learning algorithms to work properly.
- **fit_transform(Y)**: This method fits the **LabelEncoder** to the target variable **Y** and then transforms it. It returns the transformed labels. **fit_transform()** method fits the **LabelEncoder** to the target variable **Y** and then transforms it. The fitting process learns the mapping of categories to integers, and the transformation process applies this mapping to convert the categorical labels into numerical ones.
- **Y**: It represents the target variable, typically the output or dependent variable in a machine learning problem. After the transformation, **Y** contains numerical labels instead of categorical ones.

```
print(Y)
```

Output:

```
[0 1 0 0 1 1 0 1 0 1]
```