```
# print first 5 rows of X after binarization
binarize_X[0:5,:]
'Output':
array([[1., 1., 0., 0.],
       [1., 1., 0., 0.],
       [1., 1., 0., 0.],
       [1., 1., 0., 0.],
       [1., 1., 0., 0.]])
```

Encoding Categorical Variables

Most machine learning algorithms do not compute with non-numerical or categorical variables. Hence, encoding categorical variables is the technique for converting non-numerical features with labels into a numerical representation for use in machine learning modeling. Scikit-learn provides modules for encoding categorical variables including the **LabelEncoder** for encoding labels as integers, **OneHotEncoder** for converting categorical features into a matrix of integers, and **LabelBinarizer** for creating a one-hot encoding of target labels.

LabelEncoder is typically used on the target variable to transform a vector of hashable categories (or labels) into an integer representation by encoding label with values between 0 and the number of categories minus 1. This is further illustrated in Figure 18-1.

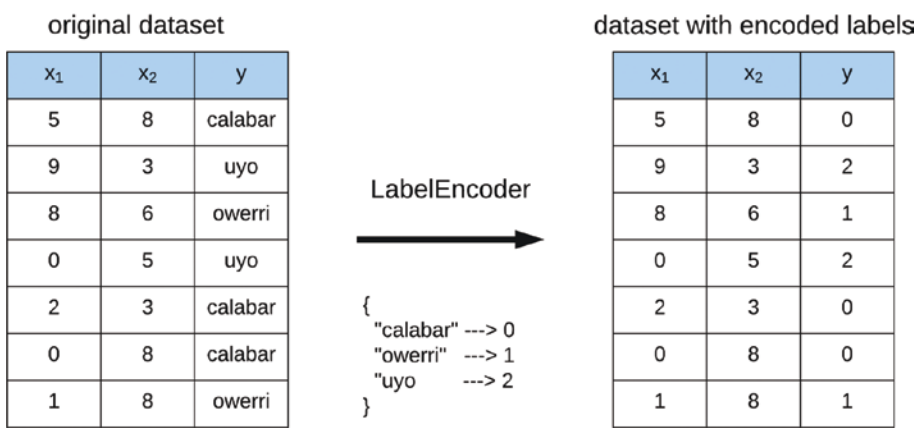
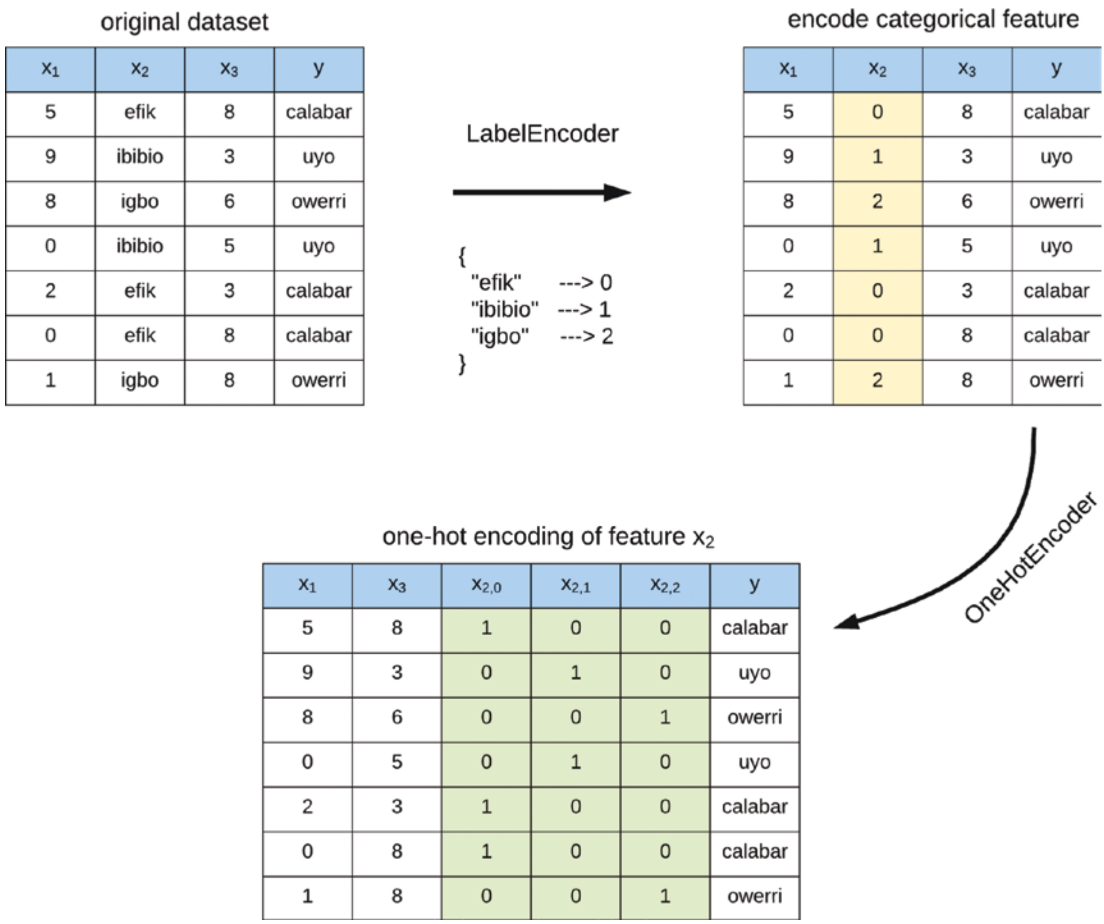


Figure 18-1. *LabelEncoder*

Let's see an example of **LabelEncoder**.

```
# import packages
from sklearn.preprocessing import LabelEncoder

# create dataset
data = np.array([[5,8,"calabar"],[9,3,"uyo"],[8,6,"owerri"],
                 [0,5,"uyo"],[2,3,"calabar"],[0,8,"calabar"],
                 [1,8,"owerri"]])
data
'Output':
array([['5', '8', 'calabar'],
       ['9', '3', 'uyo'],
       ['8', '6', 'owerri'],
       ['0', '5', 'uyo'],
       ['2', '3', 'calabar'],
       ['0', '8', 'calabar'],
       ['1', '8', 'owerri']], dtype='<U21')

# separate features and target
X = data[:,:2]
y = data[:,-1]

# encode y
encoder = LabelEncoder()
encode_y = encoder.fit_transform(y)

# adjust dataset with encoded targets
data[:,-1] = encode_y
data
'Output':
array([['5', '8', '0'],
       ['9', '3', '2'],
       ['8', '6', '1'],
       ['0', '5', '2'],
       ['2', '3', '0'],
       ['0', '8', '0'],
       ['1', '8', '1']], dtype='<U21')
```

OneHotEncoder is used to transform a categorical feature variable in a matrix of integers. This matrix is a sparse matrix with each column corresponding to one possible value of a category. This is further illustrated in Figure 18-2.



x_1	x_2	x_3	y
5	efik	8	calabar
9	ibibio	3	uyo
8	igbo	6	owerri
0	ibibio	5	uyo
2	efik	3	calabar
0	efik	8	calabar
1	igbo	8	owerri

x_1	x_2	x_3	y
5	0	8	calabar
9	1	3	uyo
8	2	6	owerri
0	1	5	uyo
2	0	3	calabar
0	0	8	calabar
1	2	8	owerri

x_1	x_3	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	y
5	8	1	0	0	calabar
9	3	0	1	0	uyo
8	6	0	0	1	owerri
0	5	0	1	0	uyo
2	3	1	0	0	calabar
0	8	1	0	0	calabar
1	8	0	0	1	owerri

Figure 18-2. *OneHotEncoder*

Let's see an example of **OneHotEncoder**.

```
# import packages
from sklearn.preprocessing import OneHotEncoder
```

```
# create dataset
data = np.array([[5,"efik", 8,"calabar"],[9,"ibibio",3,"uyo"],[8,"igbo",
6,"owerri"],[0,"ibibio",5,"uyo"],[2,"efik",3,"calabar"],[0,"efik",
8,"calabar"],[1,"igbo",8,"owerri"]])

# separate features and target
X = data[:,:3]
y = data[:,-1]

# print the feature or design matrix X
X
'Output':
array([['5', 'efik', '8'],
       ['9', 'ibibio', '3'],
       ['8', 'igbo', '6'],
       ['0', 'ibibio', '5'],
       ['2', 'efik', '3'],
       ['0', 'efik', '8'],
       ['1', 'igbo', '8']], dtype='<U21')

# one_hot_encode X
one_hot_encoder = OneHotEncoder(handle_unknown='ignore')
encode_categorical = X[:,1].reshape(len(X[:,1]), 1)
one_hot_encode_X = one_hot_encoder.fit_transform(encode_categorical)

# print one_hot encoded matrix - use todense() to print sparse matrix
# or convert to array with toarray()
one_hot_encode_X.todense()
'Output':
matrix([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.],
        [0., 1., 0.],
        [1., 0., 0.],
        [1., 0., 0.],
        [0., 0., 1.]])
```

```
# remove categorical label
X = np.delete(X, 1, axis=1)
# append encoded matrix
X = np.append(X, one_hot_encode_X.toarray(), axis=1)
X
'Output':
array([['5', '8', '1.0', '0.0', '0.0'],
       ['9', '3', '0.0', '1.0', '0.0'],
       ['8', '6', '0.0', '0.0', '1.0'],
       ['0', '5', '0.0', '1.0', '0.0'],
       ['2', '3', '1.0', '0.0', '0.0'],
       ['0', '8', '1.0', '0.0', '0.0'],
       ['1', '8', '0.0', '0.0', '1.0']], dtype='<U32')
```

Input Missing Data

It is often the case that a dataset contains several missing observations. Scikit-learn implements the **Imputer** module for completing missing values.

```
# import packages
from sklearn. impute import SimpleImputer

# create dataset
data = np.array([[5,np.nan,8],[9,3,5],[8,6,4],
                 [np.nan,5,2],[2,3,9],[np.nan,8,7],
                 [1,np.nan,5]])
data
'Output':
array([[ 5., nan,  8.],
       [ 9.,  3.,  5.],
       [ 8.,  6.,  4.],
       [nan,  5.,  2.],
       [ 2.,  3.,  9.],
       [nan,  8.,  7.],
       [ 1., nan,  5.]])
```