Encoding nominal categories features

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer #transforming categorical data into a numerical format

feature=np.array([                  #This section creates a NumPy array named feature. It's a 2D array with a single column
    ["Texas"],
    ["California"],
    ["Texas"],
    ["Delaware"],
    ["Texas"],
])

one_hot=LabelBinarizer()     #helps convert categorical data into one-hot encoded binary arrays.
one_hot.fit_transform(feature)     # fits the binarizer to the unique categories present in the feature array and transforms the data into a
```

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

```
one_hot.classes_   #one_hot.classes_ helps you understand which number stands for which category, making it easier to work with the data.
```

```
array(['California', 'Delaware', 'Texas'], dtype='<U10')
```

```
one_hot.inverse_transform(one_hot.transform(feature)) #one_hot.inverse_transform(one_hot.transform(feature)) helps you convert your data bac
```

```
array(['Texas', 'California', 'Texas', 'Delaware', 'Texas'], dtype='<U10')
```

```
import pandas as pd
pd.get_dummies(feature[:,0]) #is used to convert categorical variables into a numerical format (one-hot encoding)
```

	California	Delaware	Texas
0	False	False	True
1	True	False	False
2	False	False	True
3	False	True	False
4	False	False	True

```
multiclass_feature=[   #each element is a tuple representing a sample or observation
    ("Texas","Florida"),
    ("California","Alabama"),
    ("Texas","Florida"),
    ("Delaware","Florida"),
    ("Texas","Alabama"),
]

one_hot_multiclass=MultiLabelBinarizer()  #MultiLabelBinarizer is specifically designed for handling multi-label classification tasks

one_hot_multiclass.fit_transform(multiclass_feature)  #it identifies all the unique states across all samples.
```

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

```
one_hot_multiclass.classes_  #each unique category represents a distinct label that was present in the original data
```

```
array(['Alabama', 'California', 'Delaware', 'Florida', 'Texas'],
      dtype=object)
```

Encoding Ordinal Categorical Features

```
import pandas as pd
df=pd.DataFrame({"Score": ["Low","Low","Medium","Medium","High"]})

scale_mapper={
    "Low":1,
    "Medium":2,
    "High":3
}
df["Score"].replace(scale_mapper)
```

```
0    1
1    1
2    2
3    2
4    3
Name: Score, dtype: int64
```

Encoding Dictionary of features

```
from sklearn.feature_extraction import DictVectorizer

data_dict=[
    {"Red":2,"Blue":4},
    {"Red":4,"Blue":3},
    {"Red":1,"Yellow":2},
    {"Red":2,"Yellow":2}
]
dictvectorizer=DictVectorizer(sparse=False)
features=dictvectorizer.fit_transform(data_dict)
feature
```

```
array([['Texas'],
       ['California'],
       ['Texas'],
       ['Delaware'],
       ['Texas']], dtype='<U10')
```

```
dictvectorizer.get_feature_names_out()
```

```
array(['Blue', 'Red', 'Yellow'], dtype=object)
```

Imputing missing class values

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

X=np.array([[0,2.10,1.45],
            [1,1.18,1.33],
            [0,1.22,1.27],
            [1,-0.21,-1.19]])

X_with_nan=np.array([[np.nan,0.87,1.31],
                     [np.nan,-0.67,-0.22]])

clf=KNeighborsClassifier(3,weights='distance')
trained_model=clf.fit(X[:,1:], X[:,0])
imputed_values=trained_model.predict(X_with_nan[:,1:])
X_with_imputed=np.hstack((imputed_values.reshape(-1,1),X_with_nan[:,1:]))
np.vstack((X_with_imputed,X))
```

```
array([[ 0.  ,  0.87,  1.31],
       [ 1.  , -0.67, -0.22],
       [ 0.  ,  2.1 ,  1.45],
       [ 1.  ,  1.18,  1.33],
       [ 0.  ,  1.22,  1.27],
       [ 1.  , -0.21, -1.19]])
```

```
! pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
```

```
from sklearn.impute import SimpleImputer
X_complete =np.vstack((X_with_nan,X))
imputer=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
imputer.fit_transform(X_complete)
```

```
array([[ 0.  ,  0.87,  1.31],
       [ 0.  , -0.67, -0.22],
       [ 0.  ,  2.1 ,  1.45],
       [ 1.  ,  1.18,  1.33],
       [ 0.  ,  1.22,  1.27],
       [ 1.  , -0.21, -1.19]])
```

Handling Imbalanced Classes

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris

iris=load_iris()
features=iris.data
target=iris.target
features=features[40:,:]
target=target[40:]
target=np.where((target == 0),0,1)
target
```

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

```
weights={0:.9,1:0.1}
RandomForestClassifier(class_weight=weights)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight={0: 0.9, 1: 0.1})
```

```
RandomForestClassifier(class_weight="balanced")
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced')
```

```
i_class0=np.where(target==0)[0]
i_class1=np.where(target==1)[0]

n_class0=len(i_class0)
n_class1=len(i_class1)
i_class1_downsampled=np.random.choice(i_class1,size=n_class0,replace=False)
np.hstack((target[i_class0], target[i_class1_downsampled]))
```

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

```
np.vstack((features[i_class0,:],features[i_class1_downsampled,:]))[0:5]
```

```
array([[5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4]])
```

```
i_class0_upsampled=np.random.choice(i_class0, size=n_class1, replace=True)
np.concatenate((target[i_class0_upsampled], target[i_class1]))
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1])
```

```
np.vstack((features[i_class0_upsampled,:], features[i_class1,:]))[0:5]
```

```
     array([[5.1, 3.8, 1.6, 0.2],
            [5.1, 3.8, 1.6, 0.2],
            [5.1, 3.8, 1.6, 0.2],
            [4.4, 3.2, 1.3, 0.2],
            [4.4, 3.2, 1.3, 0.2]])
```

Encoding using Dataset(iris)

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

# Load iris dataset into a DataFrame
df_iris = pd.DataFrame(load_iris().data, columns=load_iris().feature_names)
# Convert DataFrame to NumPy array
iris_array = df_iris.values
# Display the first few rows of the DataFrame
print(df_iris.head(2))
# Display the NumPy array
print(iris_array)
```

```
       [6.7 3.3 5.7 2.5]
       [6.7 3.  5.2 2.3]
       [6.3 2.5 5.  1.9]
       [6.5 3.  5.2 2. ]
       [6.2 3.4 5.4 2.3]
       [5.9 3.  5.1 1.8]]
```

```
features = np.array(iris_array)
features
```

```
            [5.8, 2.6, 4. , 1.2],
            [5. , 2.3, 3.3, 1. ],
            [5.6, 2.7, 4.2, 1.3],
            [5.7, 3. , 4.2, 1.2],
            [5.7, 2.9, 4.2, 1.3],
            [6.2, 2.9, 4.3, 1.3],
            [5.1, 2.5, 3. , 1.1],
            [5.7, 2.8, 4.1, 1.3],
            [6.3, 3.3, 6. , 2.5],
            [5.8, 2.7, 5.1, 1.9],
            [7.1, 3. , 5.9, 2.1],
            [6.3, 2.9, 5.6, 1.8],
            [6.5, 3. , 5.8, 2.2],
            [7.6, 3. , 6.6, 2.1],
            [4.9, 2.5, 4.5, 1.7],
            [7.3, 2.9, 6.3, 1.8],
            [6.7, 2.5, 5.8, 1.8],
            [7.2, 3.6, 6.1, 2.5],
            [6.5, 3.2, 5.1, 2. ],
            [6.4, 2.7, 5.3, 1.9],
            [6.8, 3. , 5.5, 2.1],
            [5.7, 2.5, 5. , 2. ],
            [5.8, 2.8, 5.1, 2.4],
            [6.4, 3.2, 5.3, 2.3],
            [6.5, 3. , 5.5, 1.8],
            [7.7, 3.8, 6.7, 2.2],
            [7.7, 2.6, 6.9, 2.3],
            [6. , 2.2, 5. , 1.5],
            [6.9, 3.2, 5.7, 2.3],
            [5.6, 2.8, 4.9, 2. ],
            [7.7, 2.8, 6.7, 2. ],
            [6.3, 2.7, 4.9, 1.8],
            [6.7, 3.3, 5.7, 2.1],
            [7.2, 3.2, 6. , 1.8],
            [6.2, 2.8, 4.8, 1.8],
            [6.1, 3. , 4.9, 1.8],
            [6.4, 2.8, 5.6, 2.1],
            [7.2, 3. , 5.8, 1.6],
            [7.4, 2.8, 6.1, 1.9],
            [7.9, 3.8, 6.4, 2. ],
            [6.4, 2.8, 5.6, 2.2],
            [6.3, 2.8, 5.1, 1.5],
            [6.1, 2.6, 5.6, 1.4],
            [7.7, 3. , 6.1, 2.3],
            [6.3, 3.4, 5.6, 2.4],
            [6.4, 3.1, 5.5, 1.8],
            [6. , 3. , 4.8, 1.8],
            [6.9, 3.1, 5.4, 2.1],
            [6.7, 3.1, 5.6, 2.4],
            [6.9, 3.1, 5.1, 2.3],
            [5.8, 2.7, 5.1, 1.9],
            [6.8, 3.2, 5.9, 2.3],
            [6.7, 3.3, 5.7, 2.5],
            [6.7, 3. , 5.2, 2.3],
            [6.3, 2.5, 5. , 1.9],
            [6.5, 3. , 5.2, 2. ],
            [6.2, 3.4, 5.4, 2.3],
            [5.9, 3. , 5.1, 1.8]])
```

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer

multi_label = MultiLabelBinarizer()
multi_label.fit_transform(features)
```

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

```
multi_label.classes_
```

```
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6,
       1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9,
       3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2,
       4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5,
       5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8,
       6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.6, 7.7, 7.9], dtype=object)
```

```
import pandas as pd
from sklearn.datasets import load_iris

# Load Iris dataset into a DataFrame
iris = load_iris()
df_iris = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df_iris["Species"] = iris.target

# Define mapper
species_mapper = {
    0: "setosa",
    1: "versicolor",
    2: "virginica"
}

# Replace 'Species' values with mapped values
df_iris["Species"].replace(species_mapper)
df_iris["Species"].head(120)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
115    2
116    2
117    2
118    2
119    2
Name: Species, Length: 120, dtype: int64
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.feature_extraction import DictVectorizer

# Load Iris dataset into a DataFrame
iris = load_iris()
df_iris = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df_iris["Species"] = iris.target

# Convert DataFrame to a list of dictionaries
data_dict = df_iris.to_dict(orient='records')

# Create a dictionary vectorizer
dictvectorizer = DictVectorizer(sparse=False)

# Convert dictionary to feature matrix
features = dictvectorizer.fit_transform(data_dict)

print(features)
```

```
[2.  6.1 2.5 7.2 3.6]
[2.  5.1 2.  6.5 3.2]
[2.  5.3 1.9 6.4 2.7]
[2.  5.5 2.1 6.8 3. ]
[2.  5.  2.  5.7 2.5]
[2.  5.1 2.4 5.8 2.8]
[2.  5.3 2.3 6.4 3.2]
[2.  5.5 1.8 6.5 3. ]
[2.  6.7 2.2 7.7 3.8]
[2.  6.9 2.3 7.7 2.6]
[2.  5.  1.5 6.  2.2]
[2.  5.7 2.3 6.9 3.2]
[2.  4.9 2.  5.6 2.8]
[2.  6.7 2.  7.7 2.8]
[2.  4.9 1.8 6.3 2.7]
[2.  5.7 2.1 6.7 3.3]
[2.  6.  1.8 7.2 3.2]
[2.  4.8 1.8 6.2 2.8]
[2.  4.9 1.8 6.1 3. ]
[2.  5.6 2.1 6.4 2.8]
[2.  5.8 1.6 7.2 3. ]
[2.  6.1 1.9 7.4 2.8]
[2.  6.4 2.  7.9 3.8]
[2.  5.6 2.2 6.4 2.8]
[2.  5.1 1.5 6.3 2.8]
[2.  5.6 1.4 6.1 2.6]
[2.  6.1 2.3 7.7 3. ]
[2.  5.6 2.4 6.3 3.4]
[2.  5.5 1.8 6.4 3.1]
[2.  4.8 1.8 6.  3. ]
[2.  5.4 2.1 6.9 3.1]
[2.  5.6 2.4 6.7 3.1]
[2.  5.1 2.3 6.9 3.1]
[2.  5.1 1.9 5.8 2.7]
[2.  5.9 2.3 6.8 3.2]
[2.  5.7 2.5 6.7 3.3]
[2.  5.2 2.3 6.7 3. ]
[2.  5.  1.9 6.3 2.5]
[2.  5.2 2.  6.5 3. ]
[2.  5.4 2.3 6.2 3.4]
[2.  5.1 1.8 5.9 3. ]]
```

Start coding or generate with AI.