for iris

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer
feature= np.array([
    ["Texas"],
    ["California"],
     ["Texas"],
      ["Delaware"],
       ["Texas"]
    ])
# create one-hot encoder
one_hot = LabelBinarizer()
# one-hot encode feature
one_hot.fit_transform(feature)
```

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

```
one_hot.classes_
```

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

```
one_hot.inverse_transform (one_hot.transform(feature))
```

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

```
import pandas as pd
pd.get_dummies (feature[:, 0])
```

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

```
# create multiclass feature
multiclass_feature = [ ("Texas", "Florida"),
 ("California", "Alabama"),
 ("Texas", "Florida"),
  ("Delaware", "Florida"),
  ("Texas", "Alabama")
  ]
# create multiclass one-hot encoder
one_hot_multiclass = MultiLabelBinarizer()
# one-hot encode multiclass feature
one_hot_multiclass.fit_transform (multiclass_feature)
```

```
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_
```

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

```
import pandas as pd
# create features
df = pd.DataFrame({"Score": ["Low", "Low", "Medium", "Medium", "High"]})
# create mapper
scale_mapper = {
"Low": 1,
"Medium": 2,
"High": 3
}
# replace feature values with scale
df ["Score"].replace(scale_mapper)
```

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

```
from sklearn.feature_extraction import DictVectorizer
data_dict = [
{"Red": 2, "Blue": 4},
 {"Red": 4, "Blue": 3},
{"Red": 1, "Yellow":2},
{"Red": 2, "Yellow":2}
]
# create dictionary vectorizer
dictvectorizer = DictVectorizer (sparse=False) #force DictVectorizer to output a
# convert dictionary to feature matrix
features =dictvectorizer.fit_transform(data_dict)
features
```

```
array([[4., 2., 0.],
       [3., 4., 0.],
       [0., 1., 2.],
       [0., 2., 2.]])
```

```
dictvectorizer.get_feature_names_out()
```

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

```
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]])
# train KNN learner
clf = KNeighborsClassifier (3, weights='distance')
trained_model = clf.fit(X[:,1:], X[:, 0])
# predict missing values' class
imputed_values = trained_model.predict(X_with_nan [:, 1:])
# join column of predicted class with their other features
X_with_imputed = np.hstack ((imputed_values.reshape(-1, 1), X_with_nan[:, 1:]))
# join two feature matricies
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 (
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
```

```
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]])
```

---------Encoding----------^ |

Handling Imbalance Data/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, 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})
```

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

n_class0 = len(i_class0)
n_class1 = len(i_class1)

print(n_class0)
print(n_class1)

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

```
10
100
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, 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, 1, 1, 1, 1, 1, 1,
       1, 1])
```

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

```
array([[5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [5. , 3.5, 1.6, 0.6],
       [5. , 3.5, 1.6, 0.6]])
```

ENcoding Using Datasets(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)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
```

```
        [5.2 3.5 1.5 0.2]
        [5.2 3.4 1.4 0.2]
        [4.7 3.2 1.6 0.2]
        [4.8 3.1 1.6 0.2]
        [5.4 3.4 1.5 0.4]
        [5.2 4.1 1.5 0.1]
        [5.5 4.2 1.4 0.2]
        [4.9 3.1 1.5 0.2]
        [5.  3.2 1.2 0.2]
        [5.5 3.5 1.3 0.2]
        [4.9 3.6 1.4 0.1]
        [4.4 3.  1.3 0.2]
        [5.1 3.4 1.5 0.2]
        [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]
        [4.8 3.  1.4 0.3]
        [5.1 3.8 1.6 0.2]
        [4.6 3.2 1.4 0.2]
        [5.3 3.7 1.5 0.2]
        [5.  3.3 1.4 0.2]
        [7.  3.2 4.7 1.4]
        [6.4 3.2 4.5 1.5]
        [6.9 3.1 4.9 1.5]
        [5.5 2.3 4.  1.3]
```

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

```
       [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(100)
```

```
0     0
1     0
2     0
3     0
4     0
     ..
95    1
96    1
97    1
98    1
99    1
Name: Species, Length: 100, 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
```

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

print(features)
```

```
[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.