In [33]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import RandomOverSampler
from sklearn.feature_selection import SelectKBest, chi2
```

In [34]:
```
# Load the data into a pandas dataframe
data = pd.read_csv('kidneydata.csv')

# Check the distribution of the target variable
counts = data['target'].value_counts()
print(counts)
```

```
0.0    79
Name: target, dtype: int64
```

In [35]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectKBest, chi2

# Load the dataset
df = pd.read_csv('kidneydata.csv')

# Check for class imbalance
print(df['target'].value_counts())
```

```
0.0    79
Name: target, dtype: int64
```

In [36]:

```
import pandas as pd
from imblearn.over_sampling import SMOTE

# Load the dataset
df = pd.read_csv('kidneyydata.csv')

# Check the class distribution of the dataset
print(df['target'].value_counts())

# Separate the features and target
X = df.drop('target', axis=1)
y = df['target']

# Check if the data is imbalanced
if y.value_counts().min() / y.value_counts().max() < 0.5:
    # Balance the dataset using SMOTE
    oversampler = SMOTE(random_state=42)
    X, y = oversampler.fit_resample(X, y)

    # Check the class distribution after balancing
    print(y.value_counts())
else:
    print("The dataset is balanced.")
```

```
0.0    79
Name: target, dtype: int64
The dataset is balanced.
```

In [37]:

```
# Importing necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Reading in the dataset
data = pd.read_csv('kidneyydata.csv')

# Creating a LabelEncoder object
le = LabelEncoder()

# Encoding the 'gravity' feature
data['gravity_encoded'] = le.fit_transform(data['gravity'])

# Encoding the 'ph' feature
data['ph_encoded'] = le.fit_transform(data['ph'])

# Encoding the 'osmo' feature
data['osmo_encoded'] = le.fit_transform(data['osmo'])

# Encoding the 'cond' feature
data['cond_encoded'] = le.fit_transform(data['cond'])

# Encoding the 'urea' feature
data['urea_encoded'] = le.fit_transform(data['urea'])

# Encoding the 'calc' feature
data['calc_encoded'] = le.fit_transform(data['calc'])

# Encoding the 'target' feature
data['target_encoded'] = le.fit_transform(data['target'])

# Printing the first 5 rows of the dataset after encoding
print(data.head())
```

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	target	gravity_encoded
0	0	1.021	4.91	725	14.0	443	2.45	0.0	16
1	1	1.017	5.74	577	20.0	296	4.49	0.0	12
2	2	1.008	7.20	321	14.9	101	2.36	0.0	3
3	3	1.011	5.51	408	12.6	224	2.15	0.0	6
4	4	1.005	6.52	187	7.5	91	1.16	0.0	0

	ph_encoded	osmo_encoded	cond_encoded	urea_encoded	calc_encoded
0	3	50	17	65	30
1	29	36	31	41	46
2	64	9	21	8	29
3	17	18	13	32	24
4	52	0	1	5	11

	target_encoded
0	0
1	0
2	0
3	0
4	0

In [38]:
```
# Importing necessary libraries
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Reading the dataset
data = pd.read_csv('kidneyydata.csv')

# Creating the MinMaxScaler object
scaler = MinMaxScaler()

# Scaling the features
data[[  'gravity', 'ph', 'osmo', 'cond', 'urea', 'calc',]] = scaler.fit_transform(data[[  'gravity', 'ph', 'os

# Printing the first 5 rows of the dataset after scaling
print(data.head())
```

```
   Unnamed: 0   gravity        ph      osmo      cond      urea      calc  \
0           0  0.457143  0.047170  0.512869  0.270517  0.709836  0.160903
1           1  0.342857  0.308176  0.371783  0.452888  0.468852  0.304869
2           2  0.085714  0.767296  0.127741  0.297872  0.149180  0.154552
3           3  0.171429  0.235849  0.210677  0.227964  0.350820  0.139732
4           4  0.000000  0.553459  0.000000  0.072948  0.132787  0.069866

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
```

In [39]:
```python
import pandas as pd

# Load CSV file
data = pd.read_csv('kidneyydata.csv')

# Convert 'target' column to float
data['target'] = data['target'].astype(float)



# ... and so on for other columns

# Save preprocessed data to a new CSV file
data.to_csv('kidneyydata.csv', index=False)
```

In [40]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
clf = DecisionTreeClassifier()
clf.fit(X, y)
```

Out[40]:
```
DecisionTreeClassifier()
```

In [41]:
```
from sklearn.feature_selection import SelectKBest, chi2
import pandas as pd

# Load data
data = pd.read_csv('kidneyydata.csv')

# Split data into X (features) and y (target)
X = data.drop('target', axis=1)
y = data['target']

# Feature selection
selector = SelectKBest(chi2, k=5)
selector.fit(X, y)
X_new = selector.transform(X)

# Print selected features
mask = selector.get_support() # list of booleans
selected_features = [] # The list of your K best features

for bool, feature in zip(mask, X.columns):
    if bool:
        selected_features.append(feature)

print(selected_features)
```

```
['ph', 'osmo', 'cond', 'urea', 'calc']
```

In [42]:
```
# Read selected features from file
with open('kidneydata.csv', 'r') as f:
    selected_features = [line.strip() for line in f]
```

In [43]:
```
print(selected_features)
```

```
5.53,907,28.4,448,1.27', '12,1.006,7.12,242,11.3,64,1.03', '13,1.007,5.35,283,9.9,147,1.47', '14,1.011,5.2
1,450,17.9,161,1.53', '15,1.018,4.9,684,26.1,284,5.09', '16,1.007,6.63,253,8.4,133,1.05', '17,1.025,6.81,94
7,32.6,395,2.03', '18,1.008,6.88,395,26.1,95,7.68', '19,1.014,6.14,565,23.6,214,1.45', '20,1.024,6.3,874,2
9.9,380,5.16', '21,1.019,5.47,760,33.8,199,0.81', '22,1.014,7.38,577,30.1,87,1.32', '23,1.02,5.96,631,11.2,
422,1.55', '24,1.023,5.68,749,29.0,239,1.52', '25,1.017,6.76,455,8.8,270,0.77', '26,1.017,7.61,527,25.8,75,
2.17', '27,1.01,6.61,225,9.8,72,0.17', '28,1.008,5.87,241,5.1,159,0.83', '29,1.02,5.44,781,29.0,349,3.04',
'30,1.017,7.92,680,25.3,282,1.06', '31,1.019,5.98,579,15.5,297,3.93', '32,1.017,6.56,559,15.8,317,5.38', '3
3,1.008,5.94,256,8.1,130,3.53', '34,1.023,5.85,970,38.0,362,4.54', '35,1.02,5.66,702,23.6,330,3.98', '36,1.
008,6.4,341,14.6,125,1.02', '37,1.02,6.35,704,24.5,260,3.46', '38,1.009,6.37,325,12.2,97,1.19', '39,1.018,
6.18,694,23.3,311,5.64', '40,1.021,5.33,815,26.0,385,2.66', '41,1.009,5.64,386,17.7,104,1.22', '42,1.015,6.
79,541,20.9,187,2.64', '43,1.01,5.97,343,13.4,126,2.31', '44,1.02,5.68,876,35.8,308,4.49', '45,1.021,5.94,7
74,27.9,325,6.96', '46,1.024,5.77,698,19.5,354,13.0', '47,1.024,5.6,866,29.5,360,5.54', '48,1.021,5.53,775,
31.2,302,6.19', '49,1.024,5.36,853,27.6,364,7.31', '50,1.026,5.16,822,26.0,301,14.34', '51,1.013,5.86,531,2
1.4,197,4.74', '52,1.01,6.27,371,11.2,188,2.5', '53,1.011,7.01,443,21.4,124,1.27', '54,1.022,6.21,442,20.6,
398,4.18', '55,1.011,6.13,364,10.9,159,3.1', '56,1.031,5.73,874,17.4,516,3.01', '57,1.02,7.94,567,19.7,212,
6.81', '58,1.04,6.28,838,14.3,486,8.28', '59,1.021,5.56,658,23.6,224,2.33', '60,1.025,5.71,854,27.0,385,7.1
8', '61,1.026,6.19,956,27.6,473,5.67', '62,1.034,5.24,1236,27.3,620,12.68', '63,1.033,5.58,1032,29.1,430,8.
94', '64,1.015,5.98,487,14.8,198,3.16', '65,1.013,5.58,516,20.8,184,3.3', '66,1.014,5.9,456,17.8,164,6.99',
'67,1.012,6.75,251,5.1,141,0.65', '68,1.025,6.9,945,33.6,396,4.18', '69,1.026,6.29,833,22.2,457,4.45', '70,
```

In [ ]: