```
#!/usr/bin/env python
# coding: utf-8

# In[16]:

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# In[17]:

import pandas as pd
df=pd.read_csv("Mymensingh.csv")
df

# In[4]:

print(df.isnull().sum)

# In[23]:

df = df.drop(columns=['ID', 'Station', 'Year', 'Month', 'A_RAIN', 'T_RAN'])

# In[24]:
```

```
X = df.drop(columns=['RAN'])
y = df['RAN']

# In[25]:

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# In[26]:

le = LabelEncoder()
y_encoded = le.fit_transform(y)

# In[27]:

X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_encoded, test_size=0.2,
random_state=42)

# In[28]:

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```
# In[29]:

y_pred = knn.predict(X_test)

# In[31]:

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# In[17]:

plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
            xticklabels=le.classes_, yticklabels=le.classes_, cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("KNN with Median Imputation")
plt.tight_layout()
plt.show()

# In[20]:

from sklearn.neighbors import KNeighborsClassifier

# Dummy classifier just to check it works
knn = KNeighborsClassifier(n_neighbors=5)
```

```
# In[21]:


knn.fit(X_train, y_train)


# In[22]:


print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))


# In[1]:


def classification_metrics_from_confusion_matrix(TP, FP, FN, TN):
    # True Positive Rate (Sensitivity, Recall)
    TPR = TP / (TP + FN) if (TP + FN) != 0 else 0

    # False Positive Rate
    FPR = FP / (FP + TN) if (FP + TN) != 0 else 0

    # False Negative Rate
    FNR = FN / (TP + FN) if (TP + FN) != 0 else 0

    # Specificity (True Negative Rate)
    SPC = TN / (FP + TN) if (FP + TN) != 0 else 0

    # Positive Predictive Value (Precision)
    PPV = TP / (TP + FP) if (TP + FP) != 0 else 0

    # Negative Predictive Value
```

```
    NPV = TN / (TN + FN) if (TN + FN) != 0 else 0

    # Prevalence
    prevalence = (TP + FN) / (TP + FP + FN + TN) if (TP + FP + FN + TN) != 0 else 0
    # False Omission Rate
    FOR = FN / (FN + TN) if (FN + TN) != 0 else 0

    return {
        "TPR": TPR,
        "FPR": FPR,
        "FNR": FNR,
        "SPC": SPC,
        "PPV": PPV,
        "NPV": NPV,
        "Prevalence": prevalence,
        "FOR": FOR
    }

# In[11]:

import numpy as np

cm = np.array([
    [23, 17, 11],
    [4, 27, 0],
    [3, 0, 50]])

def multiclass_metrics(cm):
```

```
    n_classes = cm.shape[0]
    metrics = {}

# In[12]:

for i in range(n_classes):
        TP = cm[i, i]
        FN = np.sum(cm[i, :]) - TP
        FP = np.sum(cm[:, i]) - TP
        TN = np.sum(cm) - (TP + FP + FN)

        TPR = TP / (TP + FN) if (TP + FN) != 0 else 0       # Sensitivity, Recall
        FPR = FP / (FP + TN) if (FP + TN) != 0 else 0
        FNR = FN / (TP + FN) if (TP + FN) != 0 else 0
        SPC = TN / (FP + TN) if (FP + TN) != 0 else 0       # Specificity
        PPV = TP / (TP + FP) if (TP + FP) != 0 else 0       # Precision
        NPV = TN / (TN + FN) if (TN + FN) != 0 else 0
        prevalence = (TP + FN) / np.sum(cm) if np.sum(cm) != 0 else 0
        FOR = FN / (FN + TN) if (FN + TN) != 0 else 0       # False Omission Rate

# In[13]:

metrics[f"class_{i}"] = {
    "TPR": TPR,
    "FPR": FPR,
    "FNR": FNR,
    "SPC": SPC,
    "PPV": PPV,
```

```
    "NPV": NPV,
    "Prevalence": prevalence,
    "FOR": FOR
}
    return metrics

metrics = multiclass_metrics(cm)
for cls, vals in metrics.items():
    print(f"Metrics for {cls}:")
    for metric, value in vals.items():
print(f"  {metric}: {value:.4f}")
    print()



# In[32]:

def multiclass_metrics(confusion_matrix):
    import numpy as np
    n_classes = confusion_matrix.shape[0]
    metrics = {}

    for i in range(n_classes):
        TP = confusion_matrix[i, i]
        FN = np.sum(confusion_matrix[i, :]) - TP
        FP = np.sum(confusion_matrix[:, i]) - TP
        TN = np.sum(confusion_matrix) - (TP + FP + FN)

        TPR = TP / (TP + FN) if (TP + FN) != 0 else 0
```

```
        FPR = FP / (FP + TN) if (FP + TN) != 0 else 0
        FNR = FN / (TP + FN) if (TP + FN) != 0 else 0
        SPC = TN / (FP + TN) if (FP + TN) != 0 else 0
        PPV = TP / (TP + FP) if (TP + FP) != 0 else 0
        NPV = TN / (TN + FN) if (TN + FN) != 0 else 0
        prevalence = (TP + FN) / np.sum(confusion_matrix) if np.sum(confusion_matrix) != 0 else
0
        FOR = FN / (FN + TN) if (FN + TN) != 0 else 0
        metrics[f"class_{i}"] = {
            "TPR": TPR,
            "FPR": FPR,
            "FNR": FNR,
            "SPC": SPC,
            "PPV": PPV,
            "NPV": NPV,
            "Prevalence": prevalence,
            "FOR": FOR
        }
    return metrics

# Use it like this:
metrics = multiclass_metrics(confusion_matrix)
for cls, vals in metrics.items():
    print(f"Metrics for {cls}:")
    for metric, value in vals.items():
        print(f"  {metric}: {value:.4f}")
    print()

# In[34]:
```

```
import numpy as np
from sklearn.metrics import confusion_matrix

# Define confusion matrix
cm = np.array([[23, 17, 11],
               [ 4, 27,  0],
               [ 3,  0, 50]])

# Number of classes
n_classes = cm.shape[0]

# Initialize dictionaries
metrics = {'Class': [], 'TPR': [], 'FPR': [], 'FNR': [], 'SPC': [],
           'PPV': [], 'NPV': [], 'Prevalence': [], 'FOR': []}

for i in range(n_classes):
    TP = cm[i, i]
    FP = sum(cm[:, i]) - TP
    FN = sum(cm[i, :]) - TP
    TN = cm.sum() - (TP + FP + FN)

    TPR = TP / (TP + FN) if (TP + FN) else 0  # Sensitivity, Recall
    FPR = FP / (FP + TN) if (FP + TN) else 0
    FNR = FN / (FN + TP) if (FN + TP) else 0
    SPC = TN / (TN + FP) if (TN + FP) else 0  # Specificity
    PPV = TP / (TP + FP) if (TP + FP) else 0  # Precision
    NPV = TN / (TN + FN) if (TN + FN) else 0
```

```
    Prev = (TP + FN) / cm.sum()  # Prevalence
    FOR = FN / (FN + TN) if (FN + TN) else 0  # False Omission Rate

    metrics['Class'].append(f'Class {i}')
    metrics['TPR'].append(round(TPR, 3))
    metrics['FPR'].append(round(FPR, 3))
    metrics['FNR'].append(round(FNR, 3))
    metrics['SPC'].append(round(SPC, 3))
    metrics['PPV'].append(round(PPV, 3))
    metrics['NPV'].append(round(NPV, 3))
    metrics['Prevalence'].append(round(Prev, 3))
    metrics['FOR'].append(round(FOR, 3))

# Display nicely
import pandas as pd
df_metrics = pd.DataFrame(metrics)
print(df_metrics)

# In[ ]:
```