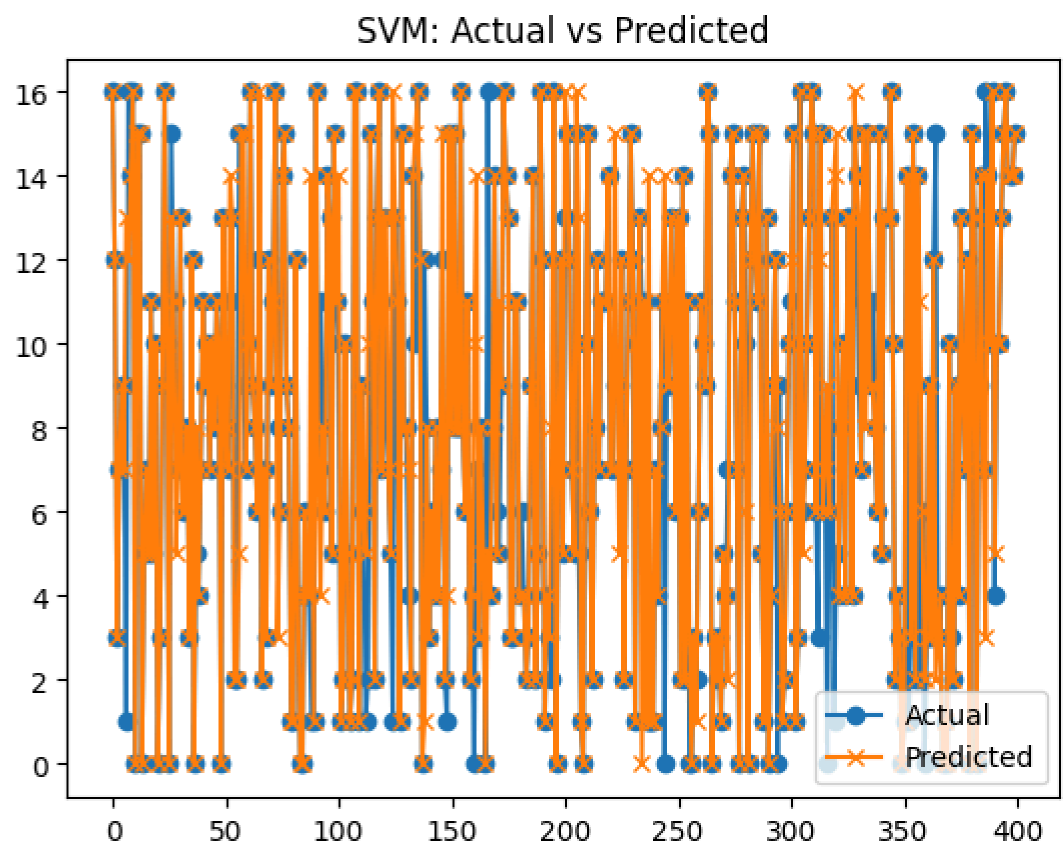
In [71]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
from sklearn.preprocessing import LabelEncoder

# Load dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # adjust t
print("First few rows of the dataset:\n", df.head())

# Encode all object (categorical) columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        print(f"Encoded column: {col}")

# Define features and target
X = df.iloc[:, :-1].values  # all columns except the last
Y = df.iloc[:, -1].values   # last column as target

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Train SVM
svm_clf = svm.SVC(kernel='rbf')
svm_clf.fit(X_train, Y_train)
svm_clf_pred = svm_clf.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(Y_test, svm_clf_pred))
print("Precision:", precision_score(Y_test, svm_clf_pred, average='weighted', ze
print("Recall:", recall_score(Y_test, svm_clf_pred, average='weighted'))
print("F1 Score:", f1_score(Y_test, svm_clf_pred, average='weighted'))
print("Confusion Matrix:\n", confusion_matrix(Y_test, svm_clf_pred))
print("Classification Report:\n", classification_report(Y_test, svm_clf_pred))

plt.plot(Y_test, label='Actual', marker='o')
plt.plot(svm_clf_pred, label='Predicted', marker='x')
plt.legend()
plt.title("SVM: Actual vs Predicted")
plt.show()
```

```
First few rows of the dataset:
   Database Fundamentals  Computer Architecture  \
0                      1                      6   
1                      2                      2   
2                      4                      4   
3                      1                      0   
4                      1                      1   

  Distributed Computing Systems  Cyber Security  Networking  \
0                             1               1           4   
1                             2               2           2   
2                             4               4           4   
3                             1               1           1   
4                             1               6           1   

  Software Development  Programming Skills  Project Management  \
0                    1                   1                   1   
1                    2                   2                   2   
2                    4                   4                   4   
3                    1                   1                   1   
4                    1                   1                   1   

  Computer Forensics Fundamentals  Technical Communication  AI ML  \
0                               1                        1      1   
1                               2                        2      2   
2                               4                        4      4   
3                               1                        1      1   
4                               1                        2      1   

  Software Engineering  Business Analysis  Communication skills  \
0                    1                  1                     1   
1                    2                  2                     2   
2                    4                  4                     5   
3                    1                  1                     6   
4                    1                  1                     1   

  Data Science  Troubleshooting skills  Graphics Designing  Role  
0            1                       1                   1     9  
1            2                       6                   2    10  
2            4                       4                   6     8  
3            1                       1                   1     4  
4            1                       1                   1     5  
Accuracy: 0.845
Precision: 0.8545785442415876
Recall: 0.845
F1 Score: 0.8454443386931196
Confusion Matrix:
 [[22  0  1  1  0  0  0  0  1  1  1  0  0  0  1  0  0]
 [ 0 19  0  0  1  1  0  0  0  0  0  0  0  1  4  0  1]
 [ 0  1 20  0  0  0  0  0  1  0  1  1  0  0  0  0  0]
 [ 0  0  1 19  0  0  0  0  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  0 20  1  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  1  0  0  1  0  1  0  1  0]
 [ 0  0  0  0  1  2 18  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  1  0  0  1  0 25  0  0  0  0  0  0  1  0  0]
 [ 0  0  0  1  0  0  0  0 20  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  0  1  1  1  0 19  0  0  0  1  0  2  0]
 [ 0  0  0  0  0  0  1  0  0  0 18  0  0  0  1  1  0]
 [ 1  0  0  0  1  0  0  0  0  1  0 22  1  0  1  1  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0 16  0  0  1  1]
```

```
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 23  0  0  1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 19  0  0]
 [ 0  0  1  0  0  1  0  0  1  0  0  0  0  1  0 25  2]
 [ 0  0  0  1  0  1  0  0  0  0  0  0  1  0  0  0 20]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.79      0.86        28
           1       0.90      0.70      0.79        27
           2       0.83      0.83      0.83        24
           3       0.86      0.90      0.88        21
           4       0.87      0.91      0.89        22
           5       0.62      0.76      0.68        17
           6       0.90      0.86      0.88        21
           7       0.93      0.89      0.91        28
           8       0.83      0.91      0.87        22
           9       0.90      0.76      0.83        25
          10       0.90      0.86      0.88        21
          11       0.92      0.79      0.85        28
          12       0.84      0.84      0.84        19
          13       0.82      0.96      0.88        24
          14       0.70      1.00      0.83        19
          15       0.81      0.81      0.81        31
          16       0.80      0.87      0.83        23

    accuracy                           0.84       400
   macro avg       0.85      0.85      0.84       400
weighted avg       0.85      0.84      0.85       400
```



In [61]:

```python
import pandas as pd
df=pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")
df.head()
```

Out[61]:

	Database Fundamentals	Computer Architecture	Distributed Computing Systems	Cyber Security	Networking	Software Development	Progra
0	1	6	1	1	4	1	
1	2	2	2	2	2	2	
2	4	4	4	4	4	4	
3	1	0	1	1	1	1	
4	1	1	1	6	1	1	

In [15]:

```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\subod\anaconda3\lib\site-pa
ckages (3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\subod\anaconda3\lib\s
ite-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\subod\anaconda3\lib\site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\subod\anaconda3\lib
\site-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\subod\anaconda3\lib
\site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\subod\anaconda3\lib\site-p
ackages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\subod\appdata\roaming
\python\python312\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\subod\appdata\roaming\python
\python312\site-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\subod\anaconda3\lib\s
ite-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\subod\appdata\roa
ming\python\python312\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\subod\appdata\roaming\python
\python312\site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

In [6]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
from sklearn.preprocessing import LabelEncoder

# Load dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # adjust t
print("First few rows of the dataset:\n", df.head())

# Encode all object (categorical) columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        print(f"Encoded column: {col}")

# Define features and target
X = df.iloc[:, :-1].values  # all columns except the last
Y = df.iloc[:, -1].values   # last column as target
```

```
# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Train SVM
svm_clf = svm.SVC(kernel='rbf')
svm_clf.fit(X_train, Y_train)
svm_clf_pred = svm_clf.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(Y_test, svm_clf_pred))
print("Precision:", precision_score(Y_test, svm_clf_pred, average='weighted', ze
print("Recall:", recall_score(Y_test, svm_clf_pred, average='weighted'))
print("F1 Score:", f1_score(Y_test, svm_clf_pred, average='weighted'))
print("Confusion Matrix:\n", confusion_matrix(Y_test, svm_clf_pred))
print("Classification Report:\n", classification_report(Y_test, svm_clf_pred))

plt.plot(Y_test, label='Actual', marker='o')[
plt.plot(svm_clf_pred, label='Predicted', marker='x')
plt.legend()
plt.title(" Actual vs Predicted")
plt.show()
```

```
First few rows of the dataset:
   Database Fundamentals  Computer Architecture  \
0               1.067293               5.991995
1               2.016854               2.091335
2               3.966334               4.012461
3               1.005729              -0.012383
4               0.944587               0.932375

   Distributed Computing Systems  Cyber Security  Networking  \
0                       1.001877        0.952914    4.025270
1                       2.027112        1.968909    1.977854
2                       4.077554        3.977973    4.001626
3                       0.950308        1.075439    1.012794
4                       0.983782        6.008291    1.101930

   Software Development  Programming Skills  Project Management  \
0              0.987561            0.998379            1.079291
1              2.037739            1.980258            1.901078
2              3.910823            4.012350            3.916494
3              0.959600            1.078096            0.993377
4              1.064748            0.975582            0.938914

   Computer Forensics Fundamentals  Technical Communication     AI ML  \
0                         1.027212                 0.960447  1.014252
1                         1.958757                 1.988026  2.034526
2                         4.080171                 3.984754  3.951388
3                         1.051688                 0.927300  0.940947
4                         1.036011                 2.040763  1.053660

   Software Engineering  Business Analysis  Communication skills  \
0              1.001678           0.961089              1.031368
1              1.943079           1.986885              2.007748
2              4.029667           3.984353              5.056547
3              1.032227           0.961934              5.991266
4              0.971050           0.990081              0.971906

   Data Science  Troubleshooting skills  Graphics Designing  Role
0      1.013526                1.008014            0.992944     9
1      2.002466                5.927714            2.036583    10
2      3.962336                4.033894            5.978927     8
3      0.991890                1.016477            0.904515     4
4      1.035123                0.993527            0.984040     5
Accuracy: 0.95
Precision: 0.9551514696253827
Recall: 0.95
F1 Score: 0.9504208077188544
Confusion Matrix:
 [[23  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 23  0  0  0  0  0  0  0  0  0  0  0  1  2  0  0]
 [ 0  0 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 21  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 23  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 17  0  0  0  0  0  1  0  0  1  0  0]
 [ 0  0  0  0  0  1 20  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 27  0  0  0  0  0  0  1  0  0]
 [ 0  0  0  0  0  0  0  0 23  0  0  0  0  0  0  1  0]
 [ 0  0  0  1  1  0  0  1  0 19  0  0  0  1  1  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 21  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 25  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0 19  0  0  0  0]
```

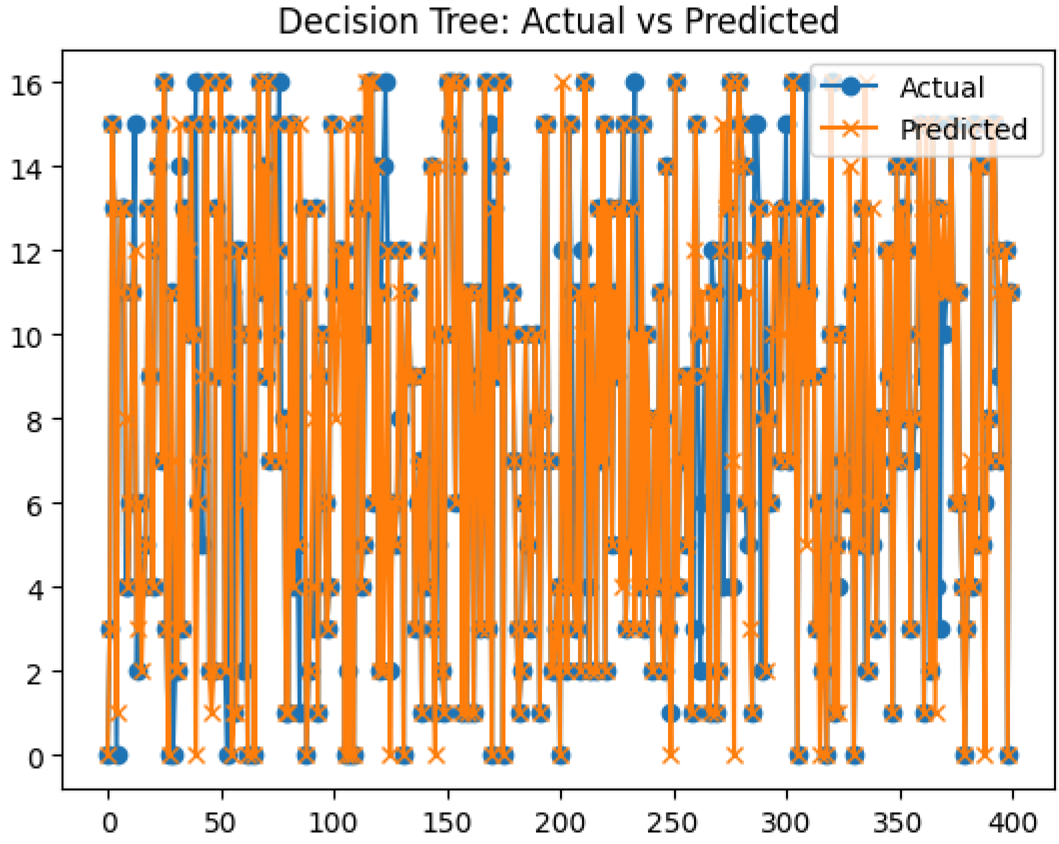
```
[ 0  0  0  0  1  0  0  0  0  0  0  0  0 25  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 20  0  0]
[ 0  0  0  0  0  2  0  0  0  0  0  0  0  0  1 27  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        24
           1       1.00      0.88      0.94        26
           2       1.00      1.00      1.00        24
           3       0.91      1.00      0.95        21
           4       0.92      0.96      0.94        24
           5       0.81      0.89      0.85        19
           6       1.00      0.95      0.98        21
           7       0.96      0.96      0.96        28
           8       1.00      0.96      0.98        24
           9       1.00      0.79      0.88        24
          10       1.00      1.00      1.00        21
          11       0.96      1.00      0.98        25
          12       1.00      0.95      0.97        20
          13       0.93      0.96      0.94        26
          14       0.77      1.00      0.87        20
          15       0.96      0.90      0.93        30
          16       1.00      1.00      1.00        23

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.96      0.95      0.95       400
```

In [73]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
import matplotlib.pyplot as plt

# Load noisy dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")

# Split features and target (assuming the last column is the target)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize and train Decision Tree
dt_clf = DecisionTreeClassifier(random_state=0)
dt_clf.fit(X_train, Y_train)
dt_pred = dt_clf.predict(X_test)

# Print performance metrics
print("Accuracy for DT:", accuracy_score(Y_test, dt_pred))
print("Precision:", precision_score(Y_test, dt_pred, average='weighted'))
print("Recall:", recall_score(Y_test, dt_pred, average='weighted'))
print("F1 Score:", f1_score(Y_test, dt_pred, average='weighted'))

# Plotting actual vs predicted
plt.plot(Y_test.values, label='Actual', marker='o')
plt.plot(dt_pred, label='Predicted', marker='x')
```

```
plt.legend()
plt.title("Decision Tree: Actual vs Predicted")
plt.show()
```

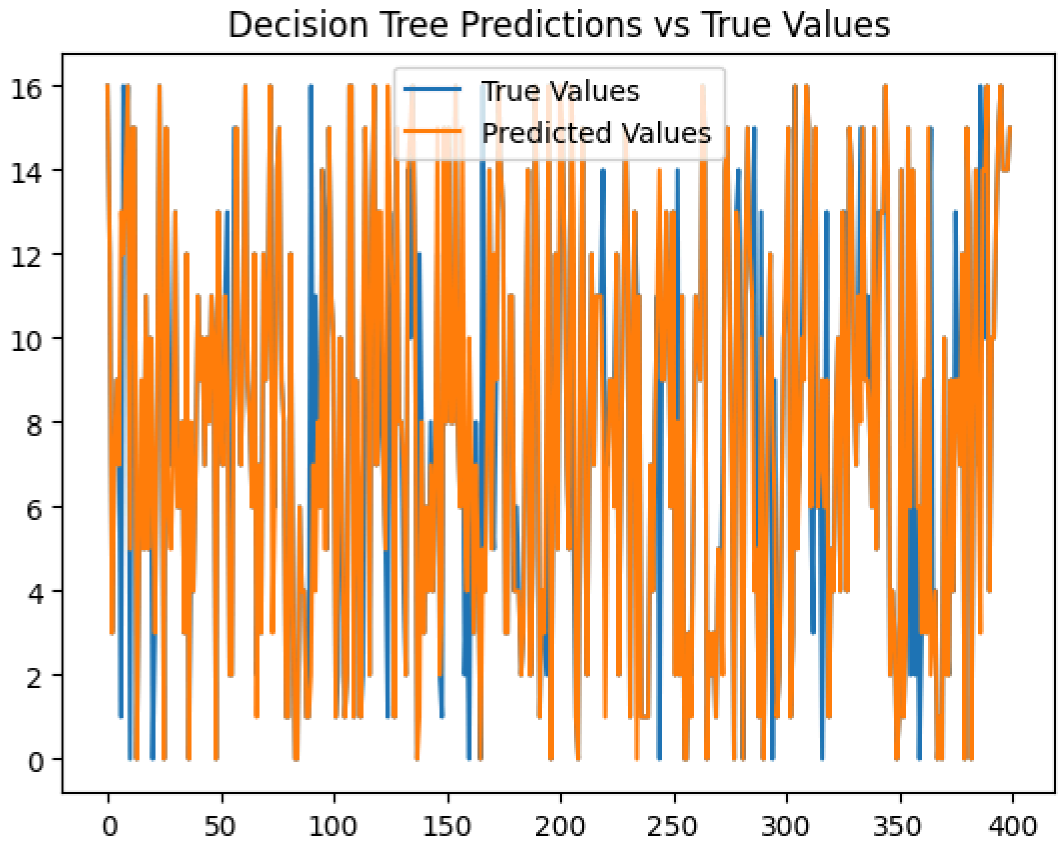
```
Accuracy for DT: 0.83
Precision: 0.8360411490038128
Recall: 0.83
F1 Score: 0.8293693361336483
```



In [75]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # Update p

# Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        print(f"Encoded column: {col}")

# Split features and target
X = df.iloc[:, :-1].values  # all columns except the last
Y = df.iloc[:, -1].values   # last column as target

# Split into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Decision Tree Classifier
dt_clf = DecisionTreeClassifier(random_state=0)
```

```
dt_clf.fit(X_train, Y_train)
dt_clf_pred = dt_clf.predict(X_test)

# Evaluation
print("Accuracy for Decision Tree:", accuracy_score(Y_test, dt_clf_pred))
print("Precision:", precision_score(Y_test, dt_clf_pred, average='weighted', zer
print("Recall:", recall_score(Y_test, dt_clf_pred, average='weighted'))
print("F1 Score:", f1_score(Y_test, dt_clf_pred, average='weighted'))
print("Confusion Matrix:\n", confusion_matrix(Y_test, dt_clf_pred))
print("Classification Report:\n", classification_report(Y_test, dt_clf_pred))

# Plotting
plt.plot(Y_test, label='True Values')
plt.plot(dt_clf_pred, label='Predicted Values')
plt.title("Decision Tree Predictions vs True Values")
plt.legend()
plt.show()
```

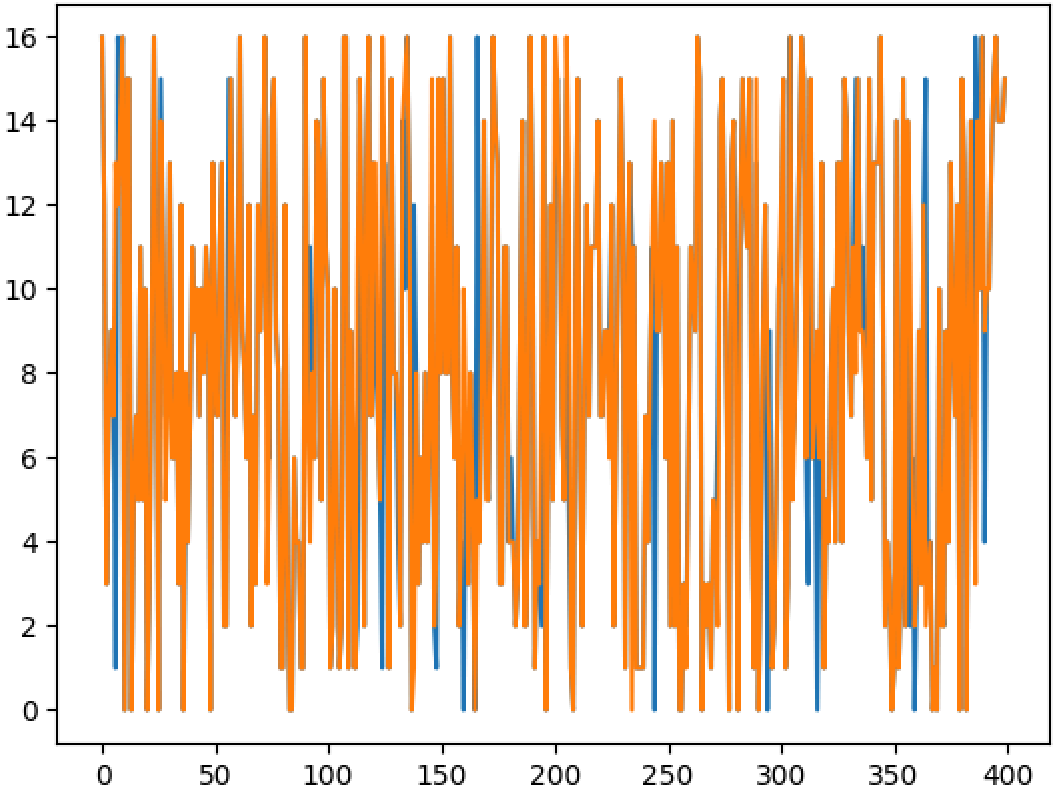
```
Accuracy for Decision Tree: 0.82
Precision: 0.8280267836611165
Recall: 0.82
F1 Score: 0.8197422224701105
Confusion Matrix:
 [[21  0  0  0  0  1  1  1  1  1  1  0  0  0  1  0  0]
 [ 0 24  0  0  1  0  0  0  0  0  0  0  0  1  0  0  1]
 [ 0  2 16  0  0  0  3  0  1  0  1  1  0  0  0  0  0]
 [ 0  0  1 19  0  0  0  0  0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0 21  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  1  0  0  0  1  1  0  0  0]
 [ 0  0  0  0  1  2 18  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  1  0  0  1  0 24  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  2 18  0  0  0  0  1  0  0  0]
 [ 0  0  0  0  0  0  2  1  0 22  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  0 17  0  0  0  2  1  0]
 [ 1  0  0  0  1  0  1  0  0  1  0 22  1  0  0  1  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0 16  0  0  2  0]
 [ 0  0  0  0  0  1  0  0  0  5  2  0  0 15  0  0  1]
 [ 0  0  0  0  0  0  0  0  2  0  0  0  1  0 16  0  0]
 [ 0  0  1  0  1  0  0  0  1  0  1  0  0  0  0 26  1]
 [ 0  0  1  1  0  1  0  0  0  0  0  0  1  0  0  0 19]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.75      0.84        28
           1       0.86      0.89      0.87        27
           2       0.80      0.67      0.73        24
           3       0.90      0.90      0.90        21
           4       0.84      0.95      0.89        22
           5       0.70      0.82      0.76        17
           6       0.69      0.86      0.77        21
           7       0.83      0.86      0.84        28
           8       0.75      0.82      0.78        22
           9       0.73      0.88      0.80        25
          10       0.77      0.81      0.79        21
          11       0.96      0.79      0.86        28
          12       0.76      0.84      0.80        19
          13       0.83      0.62      0.71        24
          14       0.84      0.84      0.84        19
          15       0.87      0.84      0.85        31
          16       0.86      0.83      0.84        23

    accuracy                           0.82       400
   macro avg       0.82      0.82      0.82       400
weighted avg       0.83      0.82      0.82       400
```



In [83]:

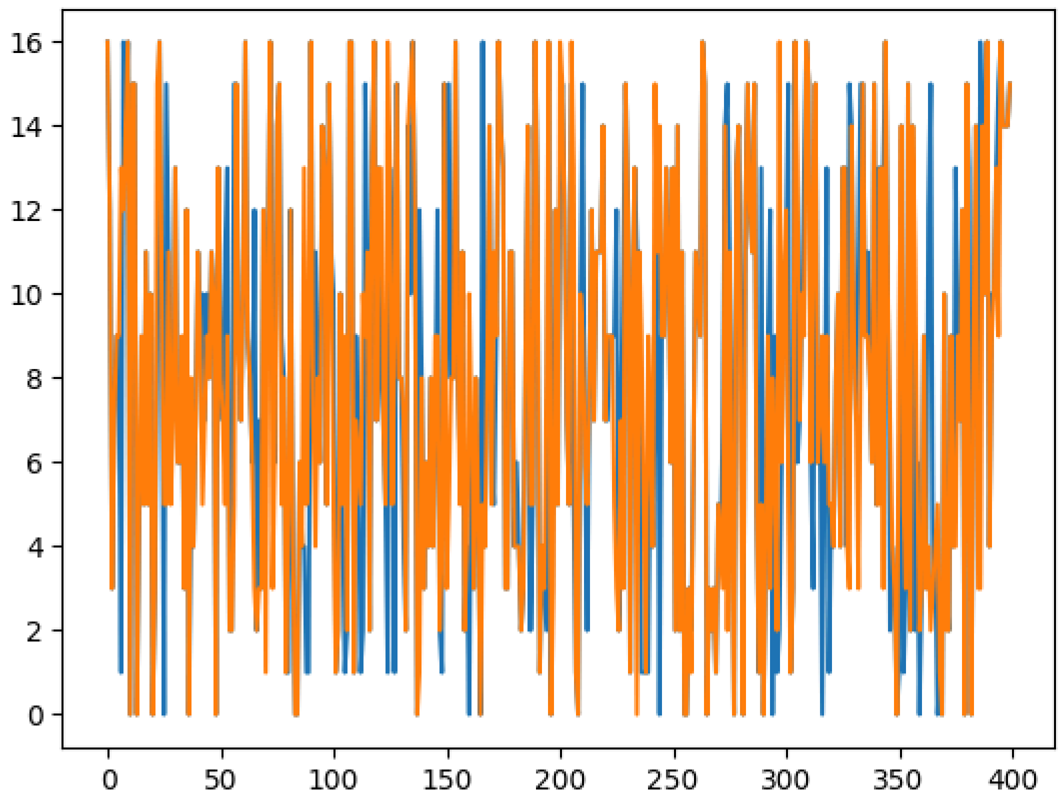
```
#logistic regression
from sklearn.linear_model import LogisticRegression

gnb_clf=LogisticRegression(solver='liblinear')
gnb_clf.fit(X_train,Y_train)
gnb_clf_pred=gnb_clf.predict(X_test)
print("accuracy for LR",accuracy_score(Y_test,gnb_clf_pred) )
print("precision",precision_score(Y_test,gnb_clf_pred,average='weighted') )
print("recall",recall_score(Y_test,gnb_clf_pred,average='weighted') )
print("f1 score",f1_score(Y_test,gnb_clf_pred,average='weighted') )
plt.plot(Y_test)
plt.plot(gnb_clf_pred)
plt.show()
```

```
accuracy for LR 0.885
precision 0.8873216419913237
recall 0.885
f1 score 0.8850840640467288
```



In [85]:
```
gnb_clf=GaussianNB()
gnb_clf.fit(X_train,Y_train)
gnb_clf_pred=gnb_clf.predict(X_test)
print("accuracy for NB",accuracy_score(Y_test,gnb_clf_pred) )
print("precision",precision_score(Y_test,gnb_clf_pred,average='weighted') )
print("recall",recall_score(Y_test,gnb_clf_pred,average='weighted') )
print("f1 score",f1_score(Y_test,gnb_clf_pred,average='weighted') )
plt.plot(Y_test)
plt.plot(gnb_clf_pred)
plt.show()
```

```
accuracy for NB 0.735
precision 0.7877820090605927
recall 0.735
f1 score 0.7432594644968857
```



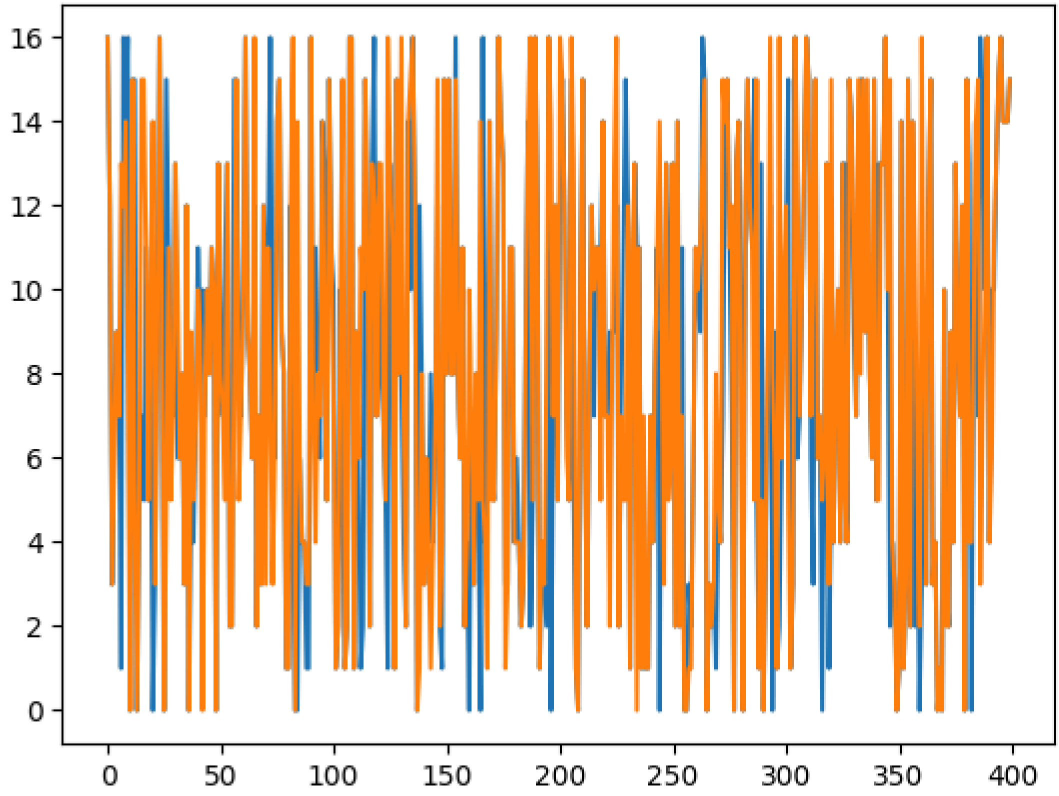
In [87]:

```
#MLP
clf=MLPClassifier(hidden_layer_sizes=(6,5),random_state=5,verbose=True,learning_
clf.fit(X_train,Y_train)
clf_pred=clf.predict(X_test)
print("accuracy for MLP",accuracy_score(Y_test,clf_pred))
plt.plot(Y_test)
plt.plot(clf_pred)
plt.show()
```

```
Iteration 1, loss = 3.23814309
Iteration 2, loss = 2.83511530
Iteration 3, loss = 2.79322793
Iteration 4, loss = 2.69215650
Iteration 5, loss = 2.58118024
Iteration 6, loss = 2.48673083
Iteration 7, loss = 2.39234743
Iteration 8, loss = 2.28922732
Iteration 9, loss = 2.18587224
Iteration 10, loss = 2.10785565
Iteration 11, loss = 2.02275975
Iteration 12, loss = 1.98217484
Iteration 13, loss = 1.93629428
Iteration 14, loss = 1.87624130
Iteration 15, loss = 1.82586455
Iteration 16, loss = 1.81208303
Iteration 17, loss = 1.79348857
Iteration 18, loss = 1.76833457
Iteration 19, loss = 1.72392761
Iteration 20, loss = 1.70205542
Iteration 21, loss = 1.71229735
Iteration 22, loss = 1.67423723
Iteration 23, loss = 1.68429065
Iteration 24, loss = 1.67145374
Iteration 25, loss = 1.63842232
Iteration 26, loss = 1.62832554
Iteration 27, loss = 1.61362539
Iteration 28, loss = 1.60565877
Iteration 29, loss = 1.58996607
Iteration 30, loss = 1.57775224
Iteration 31, loss = 1.55759856
Iteration 32, loss = 1.54986690
Iteration 33, loss = 1.53078339
Iteration 34, loss = 1.52390290
Iteration 35, loss = 1.50768629
Iteration 36, loss = 1.48725045
Iteration 37, loss = 1.48025728
Iteration 38, loss = 1.47398471
Iteration 39, loss = 1.46436769
Iteration 40, loss = 1.45357738
Iteration 41, loss = 1.44190973
Iteration 42, loss = 1.43267796
Iteration 43, loss = 1.42299294
Iteration 44, loss = 1.42216608
Iteration 45, loss = 1.42182626
Iteration 46, loss = 1.41766262
Iteration 47, loss = 1.41795623
Iteration 48, loss = 1.40240613
Iteration 49, loss = 1.39677582
Iteration 50, loss = 1.38032992
Iteration 51, loss = 1.37926191
Iteration 52, loss = 1.36393717
Iteration 53, loss = 1.36193244
Iteration 54, loss = 1.35811050
Iteration 55, loss = 1.36515781
Iteration 56, loss = 1.35439885
Iteration 57, loss = 1.36001108
Iteration 58, loss = 1.35823112
Iteration 59, loss = 1.34854732
Iteration 60, loss = 1.34700930
```

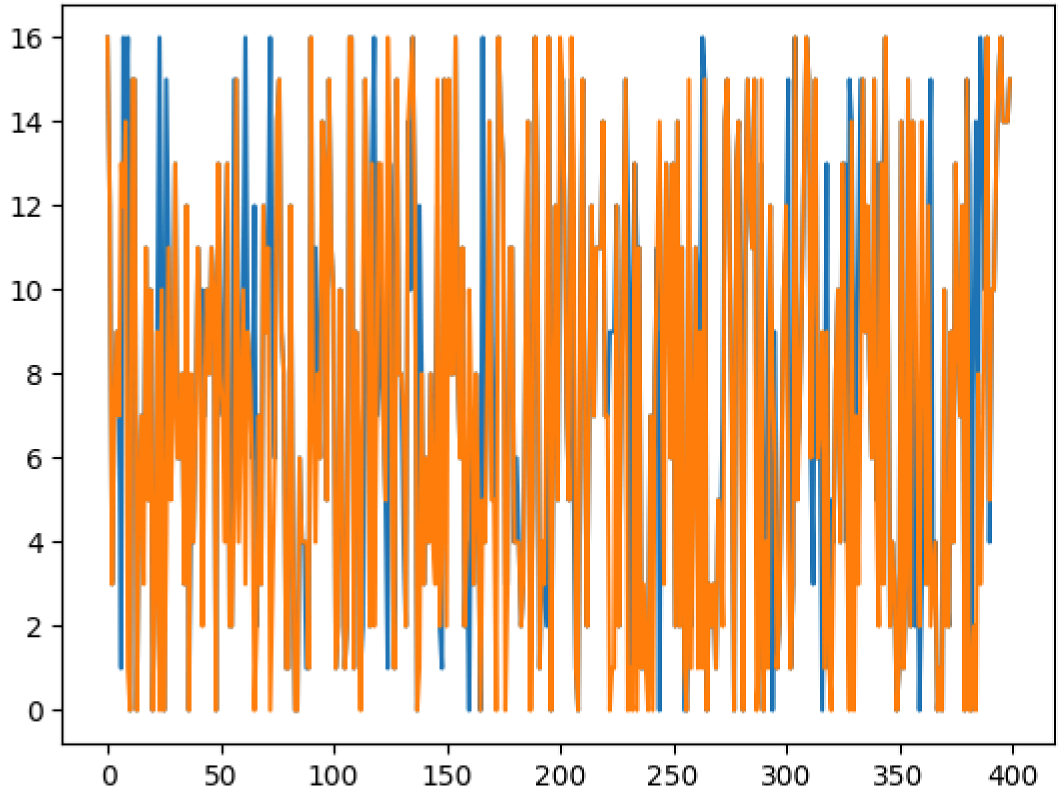
```
Iteration 61, loss = 1.34224638
Iteration 62, loss = 1.36002497
Iteration 63, loss = 1.33144315
Iteration 64, loss = 1.34159638
Iteration 65, loss = 1.33669408
Iteration 66, loss = 1.33134906
Iteration 67, loss = 1.32475983
Iteration 68, loss = 1.33512679
Iteration 69, loss = 1.31779006
Iteration 70, loss = 1.30385673
Iteration 71, loss = 1.30921505
Iteration 72, loss = 1.30595759
Iteration 73, loss = 1.30078518
Iteration 74, loss = 1.29405924
Iteration 75, loss = 1.29601844
Iteration 76, loss = 1.30068869
Iteration 77, loss = 1.28784211
Iteration 78, loss = 1.29116809
Iteration 79, loss = 1.28449247
Iteration 80, loss = 1.30049281
Iteration 81, loss = 1.30011245
Iteration 82, loss = 1.29943623
Iteration 83, loss = 1.27996184
Iteration 84, loss = 1.28116492
Iteration 85, loss = 1.27835369
Iteration 86, loss = 1.28113617
Iteration 87, loss = 1.28774506
Iteration 88, loss = 1.27355734
Iteration 89, loss = 1.28253607
Iteration 90, loss = 1.28601668
Iteration 91, loss = 1.28081770
Iteration 92, loss = 1.27826187
Iteration 93, loss = 1.26448131
Iteration 94, loss = 1.27949842
Iteration 95, loss = 1.28386054
Iteration 96, loss = 1.26835766
Iteration 97, loss = 1.26202011
Iteration 98, loss = 1.26251535
Iteration 99, loss = 1.27845538
Iteration 100, loss = 1.27312556
Iteration 101, loss = 1.27600687
Iteration 102, loss = 1.26820871
Iteration 103, loss = 1.25770053
Iteration 104, loss = 1.24879721
Iteration 105, loss = 1.24891707
Iteration 106, loss = 1.26318568
Iteration 107, loss = 1.25028913
Iteration 108, loss = 1.24680451
Iteration 109, loss = 1.24795578
Iteration 110, loss = 1.23557976
Iteration 111, loss = 1.24283755
Iteration 112, loss = 1.22220757
Iteration 113, loss = 1.22387437
Iteration 114, loss = 1.22632649
Iteration 115, loss = 1.23458252
Iteration 116, loss = 1.24214671
Iteration 117, loss = 1.23420488
Iteration 118, loss = 1.21747457
Iteration 119, loss = 1.21169409
Iteration 120, loss = 1.21966948
```

```
Iteration 121, loss = 1.21528791
Iteration 122, loss = 1.20982874
Iteration 123, loss = 1.21303602
Iteration 124, loss = 1.20919713
Iteration 125, loss = 1.21279670
Iteration 126, loss = 1.20551931
Iteration 127, loss = 1.20892897
Iteration 128, loss = 1.20007096
Iteration 129, loss = 1.19816480
Iteration 130, loss = 1.21737159
Iteration 131, loss = 1.20995720
Iteration 132, loss = 1.19418105
Iteration 133, loss = 1.20553752
Iteration 134, loss = 1.22240814
Iteration 135, loss = 1.21078708
Iteration 136, loss = 1.21652001
Iteration 137, loss = 1.19833353
Iteration 138, loss = 1.18051116
Iteration 139, loss = 1.18530979
Iteration 140, loss = 1.19551696
Iteration 141, loss = 1.19637249
Iteration 142, loss = 1.18565566
Iteration 143, loss = 1.16937364
Iteration 144, loss = 1.17345855
Iteration 145, loss = 1.17008901
Iteration 146, loss = 1.16286725
Iteration 147, loss = 1.16155155
Iteration 148, loss = 1.16799104
Iteration 149, loss = 1.16038140
Iteration 150, loss = 1.16320767
Iteration 151, loss = 1.16673185
Iteration 152, loss = 1.17612555
Iteration 153, loss = 1.16739954
Iteration 154, loss = 1.15800391
Iteration 155, loss = 1.15472512
Iteration 156, loss = 1.16805025
Iteration 157, loss = 1.16120864
Iteration 158, loss = 1.16586005
Iteration 159, loss = 1.14861584
Iteration 160, loss = 1.15924013
Iteration 161, loss = 1.15505221
Iteration 162, loss = 1.14090732
Iteration 163, loss = 1.14548166
Iteration 164, loss = 1.14796674
Iteration 165, loss = 1.13581274
Iteration 166, loss = 1.15246099
Iteration 167, loss = 1.15414458
Iteration 168, loss = 1.14341396
Iteration 169, loss = 1.14605874
Iteration 170, loss = 1.15371873
Iteration 171, loss = 1.14849626
Iteration 172, loss = 1.13803884
Iteration 173, loss = 1.14645999
Iteration 174, loss = 1.14152549
Iteration 175, loss = 1.16253156
Iteration 176, loss = 1.14385428
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. S
topping.
accuracy for MLP 0.7125
```



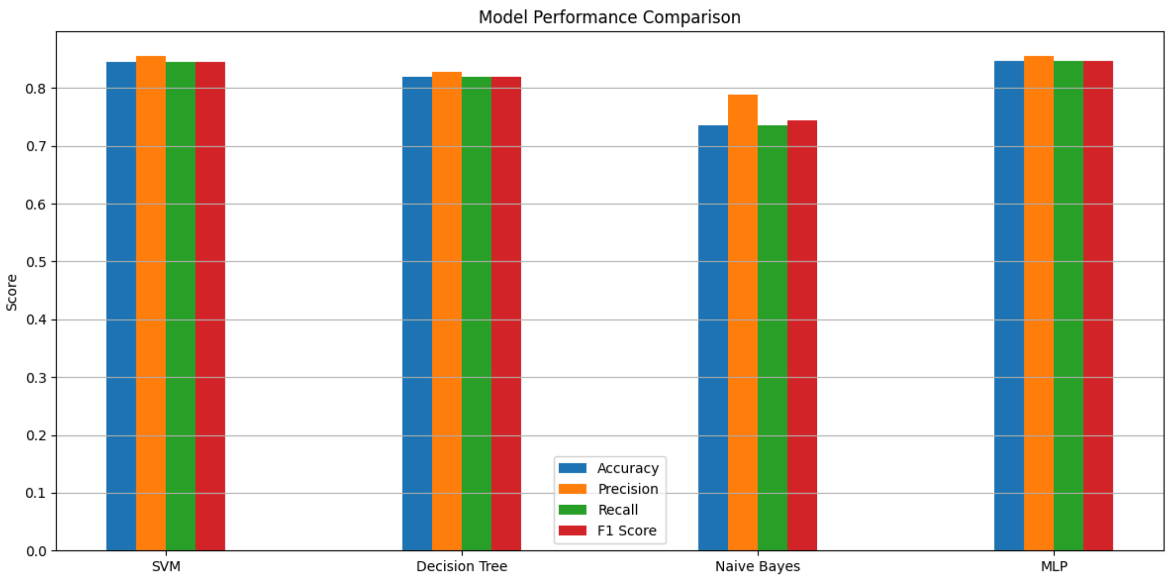
In [89]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier

gnb_clf=KNeighborsClassifier()
gnb_clf.fit(X_train,Y_train)
gnb_clf_pred=gnb_clf.predict(X_test)
print("accuracy for KNN",accuracy_score(Y_test,gnb_clf_pred) )
print("precision",precision_score(Y_test,gnb_clf_pred,average='weighted') )
print("recall",recall_score(Y_test,gnb_clf_pred,average='weighted') )
print("f1 score",f1_score(Y_test,gnb_clf_pred,average='weighted') )
plt.plot(Y_test)
plt.plot(gnb_clf_pred)
plt.show()
```

```
accuracy for KNN 0.765
precision 0.7850490238360396
recall 0.765
f1 score 0.7677185160931225
```



In [91]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Load and preprocess dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # Update t

# Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])

X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Models to evaluate
models = {
    'SVM': SVC(kernel='rbf'),
    'Decision Tree': DecisionTreeClassifier(random_state=0),
    'Naive Bayes': GaussianNB(),
    'MLP': MLPClassifier(max_iter=1000, random_state=1)
}

# Store metrics
metrics = {
    'Model': [],
```

```
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': []
}

# Evaluate each model
for name, model in models.items():
    model.fit(X_train, Y_train)
    preds = model.predict(X_test)

    metrics['Model'].append(name)
    metrics['Accuracy'].append(accuracy_score(Y_test, preds))
    metrics['Precision'].append(precision_score(Y_test, preds, average='weighted
    metrics['Recall'].append(recall_score(Y_test, preds, average='weighted'))
    metrics['F1 Score'].append(f1_score(Y_test, preds, average='weighted'))

# Convert to DataFrame for easier plotting
metrics_df = pd.DataFrame(metrics)

# Plot as grouped bar chart
x = range(len(metrics_df['Model']))
width = 0.1

plt.figure(figsize=(12, 6))
plt.bar([p - 1.5*width for p in x], metrics_df['Accuracy'], width=width, label='
plt.bar([p - 0.5*width for p in x], metrics_df['Precision'], width=width, label=
plt.bar([p + 0.5*width for p in x], metrics_df['Recall'], width=width, label='Re
plt.bar([p + 1.5*width for p in x], metrics_df['F1 Score'], width=width, label='

plt.xticks(x, metrics_df['Model'])
plt.ylabel("Score")
plt.title("Model Performance Comparison")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

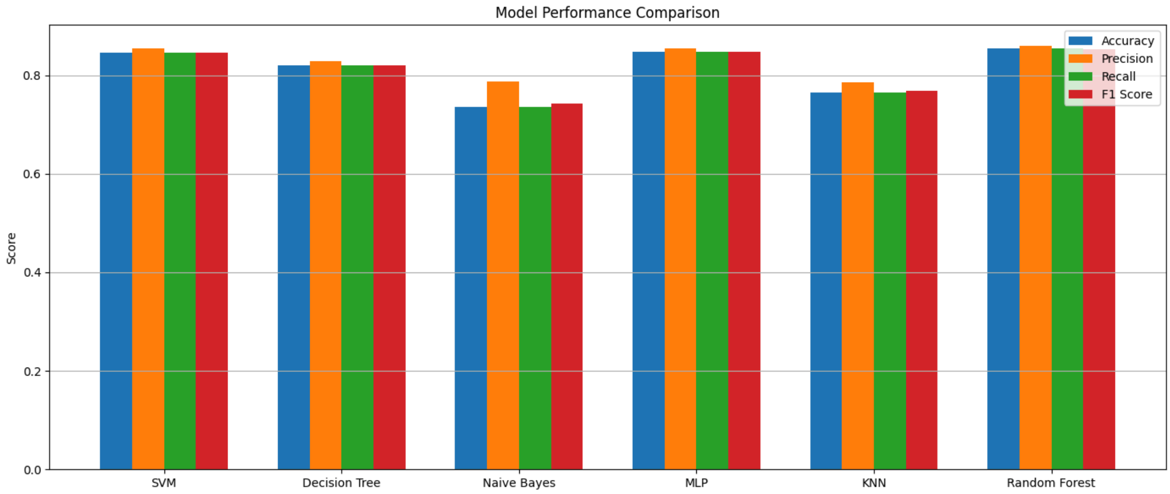


In [93]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Load and preprocess dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # Update p

# Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])

# Features and target
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

# Train/test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Models dictionary
models = {
    'SVM': SVC(kernel='rbf'),
    'Decision Tree': DecisionTreeClassifier(random_state=0),
    'Naive Bayes': GaussianNB(),
    'MLP': MLPClassifier(max_iter=1000, random_state=1),
    'KNN': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(random_state=1)
}

# Initialize metric storage
metrics = {
    'Model': [],
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': []
}

# Evaluate each model
for name, model in models.items():
    model.fit(X_train, Y_train)
    preds = model.predict(X_test)

    metrics['Model'].append(name)
    metrics['Accuracy'].append(accuracy_score(Y_test, preds))
    metrics['Precision'].append(precision_score(Y_test, preds, average='weighted
    metrics['Recall'].append(recall_score(Y_test, preds, average='weighted'))
    metrics['F1 Score'].append(f1_score(Y_test, preds, average='weighted'))

# Convert to DataFrame
metrics_df = pd.DataFrame(metrics)

# Plotting grouped bar chart
x = range(len(metrics_df['Model']))
width = 0.18
```

```
plt.figure(figsize=(14, 6))
plt.bar([p - 1.5*width for p in x], metrics_df['Accuracy'], width=width, label='
plt.bar([p - 0.5*width for p in x], metrics_df['Precision'], width=width, label=
plt.bar([p + 0.5*width for p in x], metrics_df['Recall'], width=width, label='Re
plt.bar([p + 1.5*width for p in x], metrics_df['F1 Score'], width=width, label='

plt.xticks(x, metrics_df['Model'])
plt.ylabel("Score")
plt.title("Model Performance Comparison")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



In [95]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Load dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")  # Adjust p

# Encode categorical features
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])

# Split data into features and target
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_
```

```
# Define models
models = {
    'SVM': SVC(kernel='rbf'),
    'Decision Tree': DecisionTreeClassifier(random_state=0),
    'Naive Bayes': GaussianNB(),
    'MLP': MLPClassifier(max_iter=1000, random_state=1),
    'KNN': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(random_state=1),
    'Logistic Regression': LogisticRegression(max_iter=1000)
}

# Store metrics
metrics = {
    'Model': [],
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': []
}

# Train and evaluate models
for name, model in models.items():
    model.fit(X_train, Y_train)
    preds = model.predict(X_test)

    metrics['Model'].append(name)
    metrics['Accuracy'].append(accuracy_score(Y_test, preds))
    metrics['Precision'].append(precision_score(Y_test, preds, average='weighted
    metrics['Recall'].append(recall_score(Y_test, preds, average='weighted'))
    metrics['F1 Score'].append(f1_score(Y_test, preds, average='weighted'))

# Create DataFrame
metrics_df = pd.DataFrame(metrics)

# Plot Histogram
x = range(len(metrics_df['Model']))
width = 0.18

plt.figure(figsize=(15, 6))
plt.bar([p - 1.5 * width for p in x], metrics_df['Accuracy'], width=width, label
plt.bar([p - 0.5 * width for p in x], metrics_df['Precision'], width=width, labe
plt.bar([p + 0.5 * width for p in x], metrics_df['Recall'], width=width, label='
plt.bar([p + 1.5 * width for p in x], metrics_df['F1 Score'], width=width, label

plt.xticks(x, metrics_df['Model'], rotation=15)
plt.ylabel("Score")
plt.title("Model Performance Comparison")
plt.legend()
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

In [97]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Load dataset
df = pd.read_csv("E:/Destiny/noisy_dataset_with_flipped_labels.csv")

# Encode categorical columns
for col in df.columns:
    if df[col].dtype == 'object':
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])

# Features and target
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

# Split data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

# Random Forest model
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train, Y_train)

# Predict
pred = rf.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(Y_test, pred))
print("Precision:", precision_score(Y_test, pred, average='weighted', zero_divis
print("Recall:", recall_score(Y_test, pred, average='weighted'))
print("F1 Score:", f1_score(Y_test, pred, average='weighted'))
print("\nClassification Report:\n", classification_report(Y_test, pred))
```

```
Accuracy: 0.855
Precision: 0.8590689135140223
Recall: 0.855
F1 Score: 0.8529407241097071

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.79      0.86        28
           1       0.89      0.89      0.89        27
           2       0.80      0.50      0.62        24
           3       0.90      0.90      0.90        21
           4       0.84      0.95      0.89        22
           5       0.71      0.88      0.79        17
           6       0.71      0.81      0.76        21
           7       0.89      0.89      0.89        28
           8       0.83      0.91      0.87        22
           9       0.79      0.88      0.83        25
          10       0.90      0.90      0.90        21
          11       0.96      0.86      0.91        28
          12       0.85      0.89      0.87        19
          13       0.86      0.79      0.83        24
          14       0.95      1.00      0.97        19
          15       0.82      0.87      0.84        31
          16       0.87      0.87      0.87        23

    accuracy                           0.85       400
   macro avg       0.86      0.86      0.85       400
weighted avg       0.86      0.85      0.85       400
```

In [ ]: