In [61]:

```
#
# Appendix: Python Code for Stock Movement Prediction
# Course: DSC630 - Predictive Analytics
# Author: Akshay Sharma
#

# Import necessary libraries for data manipulation, financial analysis, and machine learning
import yfinance as yf
import pandas as pd
from ta import add_all_ta_features
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix
import warnings

# Ignore warnings to keep the output clean
warnings.filterwarnings('ignore')

# 1. Download historical stock data for SPY from Yahoo Finance
# The period is set from the start of 2020 to the start of 2023.
# auto_adjust=False is used to get the raw OHLCV data without adjustments.
df = yf.download('SPY', start='2020-01-01', end='2023-01-01', auto_adjust=False)
df.reset_index(inplace=True) # Move the 'Date' from the index to a column

# 2. Flatten column names if they are a MultiIndex
# This can happen with yfinance and this step ensures column names are simple strings.
if isinstance(df.columns, pd.MultiIndex):
    df.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col for col in df.columns]

# 3. Find the actual column names for OHLCV and map them to standard names
# This makes the code more robust if yfinance changes its column naming conventions.
columns_map = {}
for base in ['Open', 'High', 'Low', 'Close', 'Volume']:
    match = [col for col in df.columns if base in col]
    if match:
        columns_map[match[0]] = base  # Rename the first found match to the standard name

# 4. Apply the renaming map to the DataFrame
df.rename(columns=columns_map, inplace=True)

# 5. Validate that all required columns are present after the renaming process
required_cols = ['Open', 'High', 'Low', 'Close', 'Volume']
missing_cols = [col for col in required_cols if col not in df.columns]
if missing_cols:
    raise ValueError(f"Required columns missing after rename: {missing_cols}")

# 6. Drop any rows that have missing values in the core OHLCV columns
df.dropna(subset=required_cols, inplace=True)

# 7. Add a comprehensive set of technical indicators using the 'ta' library
# This function automatically calculates dozens of indicators based on the OHLCV data.
# fillna=True will fill any resulting NaNs, though we will handle them again later.
df = add_all_ta_features(
    df,
    open='Open',
    high='High',
    low='Low',
    close='Close',
    volume='Volume',
    fillna=True
)

# 8. Create the binary target variable 'Target'
# '1' if the next day's close is higher than the current day's close, else '0'.
# We shift by -1 to use the *next* day's movement as the target for the *current* day's features.
df['Target'] = (df['Close'].shift(-1) > df['Close']).astype(int)
# Drop the last row which will have a NaN in the 'Target' column due to the shift
df.dropna(inplace=True)

# 9. Perform a time-aware train-test split (80% train, 20% test)
# This is crucial for time-series data to prevent data leakage from the future.
train_size = int(len(df) * 0.8)
train = df.iloc[:train_size]
test = df.iloc[train_size:]

c

# 11. Scale the features using StandardScaler
# The scaler is fit ONLY on the training data to prevent data leakage.
# Both the training and test sets are then transformed by the fitted scaler.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 12. Initialize, train, and evaluate the XGBoost Classifier model
# use_label_encoder=False and eval_metric='logloss' are set to avoid deprecation warnings.
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Train the model on the scaled training data
model.fit(X_train_scaled, y_train)

# Make predictions on the unseen, scaled test set
```

```python
y_pred = model.predict(X_test_scaled)

# Print the performance metrics
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(f"Precision: {precision_score(y_test, y_pred):.2f}")
print(f"Recall: {recall_score(y_test, y_pred):.2f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred):.2f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
[**********100%**********]  1 of 1 completed
Accuracy: 0.5
Precision: 0.4491525423728814
Recall: 0.828125
ROC-AUC: 0.5447443181818181
Confusion Matrix:
 [[23 65]
 [11 53]]
```