In [5]:
```
df['label'] = df['label'].map({'ham': 0, 'spam': 1})
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(
    df['message'], df['label'], test_size=0.2, random_state=42, strati
)
```

In [7]:
```
vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [8]:
```
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```

Out[8]:
```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [9]:
```
y_pred = model.predict(X_test_tfidf)
```

In [10]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
```

In [11]:
```
print("\nConfusion Matrix:\n", cm)

print("\nTP =", tp)
print("FP =", fp)
print("TN =", tn)
print("FN =", fn)
```

```

Confusion Matrix:
 [[966   0]
 [ 35 114]]

TP = 114
FP = 0
TN = 966
FN = 35
```

In [12]:
```
accuracy = (tp + tn) / (tp + tn + fp + fn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
```

In [13]:
```
print("\nAccuracy =", accuracy)
print("Error Rate =", error_rate)
print("Precision =", precision)
print("Recall =", recall)
```

```
Accuracy = 0.968609865470852
Error Rate = 0.03139013452914796
Precision = 1.0
Recall = 0.7651006711409396
```

In [24]:
```
X = df.iloc[:, 0:4].values
y = df[target_col].values
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

In [26]:
```
model = GaussianNB()
model.fit(X_train, y_train)
```

Out[26]:
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [27]:
```
y_pred = model.predict(X_test)
```

In [28]:
```
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
```

In [29]:
```
classes = np.unique(y_test)

print("\n----- Class-wise Metrics -----\n")

for i, cls in enumerate(classes):
    TP = cm[i, i]
    FP = cm[:, i].sum() - TP
    FN = cm[i, :].sum() - TP
    TN = cm.sum() - (TP + FP + FN)

    accuracy = (TP + TN) / cm.sum()
    error_rate = 1 - accuracy
    precision = TP / (TP + FP) if (TP + FP) != 0 else 0
    recall = TP / (TP + FN) if (TP + FN) != 0 else 0

    print(f"Class: {cls}")
    print("TP =", TP)
    print("FP =", FP)
    print("TN =", TN)
    print("FN =", FN)
    print("Accuracy =", accuracy)
    print("Error Rate =", error_rate)
    print("Precision =", precision)
    print("Recall =", recall)
    print("-----------------------------------")
```

```
----- Class-wise Metrics -----

Class: Iris-setosa
TP = 10
FP = 0
TN = 20
FN = 0
Accuracy = 1.0
Error Rate = 0.0
Precision = 1.0
Recall = 1.0
----------------------------------
Class: Iris-versicolor
TP = 9
FP = 0
TN = 20
FN = 1
Accuracy = 0.9666666666666667
Error Rate = 0.033333333333333326
Precision = 1.0
Recall = 0.9
----------------------------------
Class: Iris-virginica
TP = 10
FP = 1
TN = 19
FN = 0
Accuracy = 0.9666666666666667
Error Rate = 0.033333333333333326
Precision = 0.9090909090909091
Recall = 1.0
----------------------------------
```

In [30]:

```
overall_accuracy = np.trace(cm) / np.sum(cm)
print("\nOverall Accuracy =", overall_accuracy)
```

```
Overall Accuracy = 0.9666666666666667
```

In [ ]: