TASK 9

SUBJECT:

Programming For AI

PROGRAM:

BS DATA SCIENCE

SUBMITTED TO:

Sir Rasikh Ali

SUBMITTED BY:

FIZZA FAROOQ

ROLL NUMBER:

SU92-BSDSM-F23-017

BSDS (4A)

TASK 9:NLP

TEXT CLASSIFICATION USING NAVIE'S BYES

Naive Bayes Classification on Sales Data

Introduction

This Python script implements a Naive Bayes classification model using the GaussianNB algorithm to predict the country based on sales data. The dataset contains information about sales representatives, products, sales amounts, and shipping details. The code performs data preprocessing, model training, evaluation, and user input prediction.

import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

✓ 3.4s

- **Explanation:**
- pandas: Used for data manipulation and loading the dataset.
- numpy: Supports numerical operations.
- matplotlib.pyplot and seaborn: Used for visualizing the confusion matrix.
- sklearn.model_selection.train_test_split: Splits data into training and testing sets.
- sklearn.naive_bayes.GaussianNB: Implements the Naive Bayes classification model.
- sklearn.metrics: Used to evaluate model performance.
- sklearn.preprocessing.LabelEncoder: Encodes categorical data into numerical values.

Load dataset

```
df= pd.read_csv("data.csv")
```

[3] ✓ 0.0s

```
df.head(5)
```

[4] ✓ 0.0s

	Sales Person	Country	Product	Date	Amount	Boxes Shipped
0	Jehu Rudeforth	UK	Mint Chip Choco	04-Jan-22	$5,320	180
1	Van Tuxwell	India	85% Dark Bars	01-Aug-22	$7,896	94
2	Gigi Bohling	India	Peanut Butter Cubes	07-Jul-22	$4,501	91
3	Jan Morforth	Australia	Peanut Butter Cubes	27-Apr-22	$12,726	342
4	Jehu Rudeforth	UK	Peanut Butter Cubes	24-Feb-22	$13,685	184

```
df.tail()
```

[5] ✓ 0.0s

	Sales Person	Country	Product	Date	Amount	Boxes Shipped
1089	Karlen McCaffrey	Australia	Spicy Special Slims	17-May-22	$4,410	323
1090	Jehu Rudeforth	USA	White Choc	07-Jun-22	$6,559	119
1091	Ches Bonnell	Canada	Organic Choco Syrup	26-Jul-22	$574	217

Explanation:

- The dataset is loaded from the given file path into a DataFrame.
- data.head(): Displays the first five rows of the dataset.
- data.columns: Lists all column names.

Select features and target

```
feature_columns = ["Sales Person", "Product", "Amount", "Boxes Shipped"]
target_column = "Country"
```

[8] ✓ 0.0s

Explanation:

- feature_columns: List of columns used as input features.
- target_column: The column to be predicted (Country).

Encode categorical columns

```
encoder = LabelEncoder()
df_encoded = df.copy()
for col in feature_columns + [target_column]:
    df_encoded[col] = encoder.fit_transform(df_encoded[col])
```

[9] ✓ 0.0s

Explanation:

- LabelEncoder() converts categorical data (e.g., names, products) into numerical values.
- The loop applies encoding to all feature columns and the target column.

Split dataset

```
X = df_encoded[feature_columns]
y = df_encoded[target_column]
```

10] ✓ 0.0s

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

11] ✓ 0.0s

- X: Contains the features (Sales Person, Product, Amount, Boxes Shipped).
- y: Contains the target labels (Country).
- train_test_split: Splits the dataset into 80% training and 20% testing data.

Train Naive Bayes model

```
model = GaussianNB()
model.fit(X_train, y_train)
```

[12] ✓ 0.0s Python

```
GaussianNB
GaussianNB()
```

Explanation:

- GaussianNB(): Initializes the Naive Bayes classifier.
- fit(): Trains the model using training data.

Predictions

```
y_pred = model.predict(X_test)
```

✓ 0.0s

- predict(): Uses the trained model to predict test data labels.

Evaluate model

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')
```

[14] ✓ 0.0s

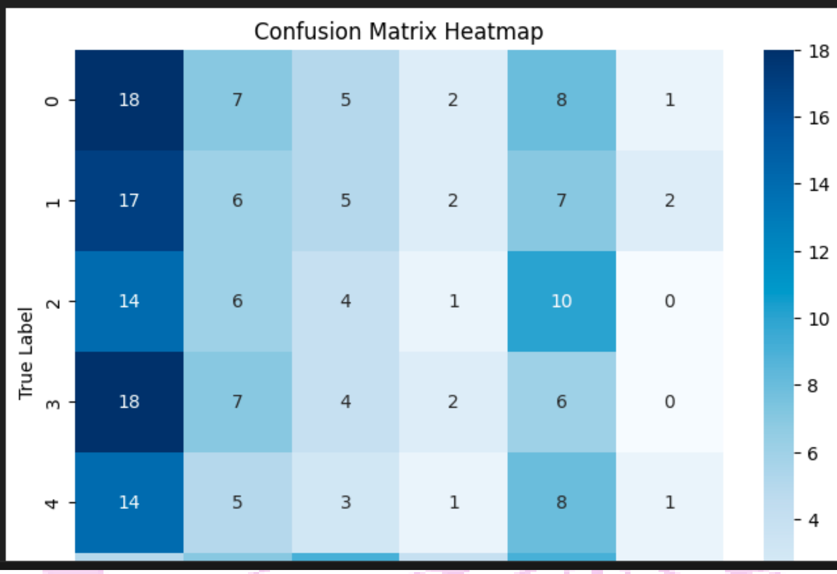
```
Accuracy: 17.81%
```

- accuracy_score(): Calculates the percentage of correctly classified instances.
- confusion_matrix(): Generates a confusion matrix.
- The accuracy percentage is printed.

Confusion matrix heatmap

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix Heatmap')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

✓ 1.0s



- `seaborn.heatmap()`: Creates a heatmap visualization of the confusion matrix.
- `plt.show()`: Displays the visualization.

Encode categorical columns

```
encoders = {}

for col in feature_columns + [target_column]:
    encoders[col] = LabelEncoder()
    df_encoded[col] = encoders[col].fit_transform(df_encoded[col])
```

✓ 0.0s

Predict on user input (Example: Assume user enters new data)

```
user_input = pd.DataFrame([["John Doe", "Laptop", 5000, 10]], columns=feature_columns)
```

✓ 0.0s

Encode user input

```
for col in feature_columns:
    if user_input[col][0] in encoders[col].classes_:
        user_input[col] = encoders[col].transform(user_input[col])
    else:
        user_input[col] = -1  # Assign unknown values a special number

predicted_country = model.predict(user_input)
print(f"The predicted country for the input data is: {encoders[target_column].inverse_transform(predicted_co
```

✓ 0.0s Python

```
The predicted country for the input data is: 3
```

Explanation:

- Creates a DataFrame with user input data.
- Encodes the user input using LabelEncoder.
- Predicts the country using predict().
- Converts the predicted numerical value back to the original country name.

Conclusion

- This script efficiently trains a Naive Bayes model on sales data and evaluates its performance using accuracy and a confusion matrix. Additionally, it provides functionality for predicting a country based on user-provided input.

Text Classification using logistic Regression

import libraries

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

✓ 0.8s

- pandas: Used for handling datasets.
- CountVectorizer: Converts text data into numerical features.
- train_test_split: Splits the dataset into training and testing subsets.
- LogisticRegression: Machine learning model for classification.

- LabelEncoder: Encodes categorical labels into numerical values.
- accuracy_score, confusion_matrix: Metrics to evaluate model performance.

```python
data = pd.read_csv("data.csv")
print("Dataset Preview:")
print(data.head())

print("\nAvailable Columns:", data.columns)
```

0.0s

```
Dataset Preview:
    Sales Person    Country             Product       Date     Amount  \
0  Jehu Rudeforth         UK      Mint Chip Choco  04-Jan-22   $5,320
1     Van Tuxwell      India       85% Dark Bars  01-Aug-22   $7,896
2    Gigi Bohling      India  Peanut Butter Cubes  07-Jul-22   $4,501
3    Jan Morforth  Australia  Peanut Butter Cubes  27-Apr-22  $12,726
4  Jehu Rudeforth         UK  Peanut Butter Cubes  24-Feb-22  $13,685
```

- Reads the dataset from a CSV file.
- Displays the column names in the dataset.

Use 'Product' as text input and 'Country' as the label

```python
text_col = 'Product'
label_col = 'Country'
if text_col not in data.columns or label_col not in data.columns:
    raise ValueError(f"Could not find appropriate label and text columns. Found: {data.columns}")

label_encoder = LabelEncoder()
data['label'] = label_encoder.fit_transform(data[label_col])
```

0.0s Python

- Uses "Product" as the feature (input text) and "Country" as the target (output category).
- Ensures that the required columns exist in the dataset.
- Converts the categorical "Country" column into numerical labels.

Feature extraction using CountVectorizer

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data[text_col])
y = data['label']
```
[40] ✓ 0.0s

Split data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```
[41] ✓ 0.0s

- Uses CountVectorizer to transform product names into a numeric format.
- Splits the dataset into **75% training** and **25% testing**.

Train Logistic Regression model

```
model = LogisticRegression(random_state=42, max_iter=200)
model.fit(X_train, y_train)
```
[42] ✓ 0.1s

```
LogisticRegression
LogisticRegression(max_iter=200, random_state=42)
```

Evaluate model accuracy

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Model Accuracy: {accuracy * 100:.2f}%')
```
[43] ✓ 0.0s

```
Model Accuracy: 16.79%
```

- Trains the `LogisticRegression` model.
- Predicts the test set and calculates accuracy.

Function to classify a new product

```python
def classify_product(model, vectorizer, label_encoder, product_name):
    product_vect = vectorizer.transform([product_name])
    predicted_label = model.predict(product_vect)[0]
    predicted_country = label_encoder.inverse_transform([predicted_label])[0]
    return predicted_country
```

[44] ✓ 0.0s

Example usage

```python
product_name = "Laptop"
predicted_country = classify_product(model, vectorizer, label_encoder, product_name)
print(f"The product '{product_name}' is most likely from '{predicted_country}'")
```

[45] ✓ 0.0s

```
The product 'Laptop' is most likely from 'Australia'
```

Text Classification using RNN

- Converts a new product name into a vector.
- Uses the trained model to predict the country.
- Decodes the prediction back to the country name.
- Predicts the country for the product **"Laptop"**.

Text Classification using RNN

Text Classification using RNN

```python
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv("data.csv")

X_text = data['Product'].astype(str)  # Convert to string to avoid dtype issues
y_labels = data['Country'].astype(str)
```

✓ 0.0s

- **TensorFlow:** Used to build and train the deep learning model.
- **Pandas & NumPy:** Handle dataset operations.
- **Matplotlib:** Visualizes training progress.
- **LabelEncoder:** Converts categorical country labels into numerical values.
- **train_test_split:** Splits the dataset into training and testing sets
- The **'Product'** column is the **feature (input text)**.
- The **'Country'** column is the **target (label to predict)**.

Encode country labels as numbers

```python
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y_labels)
```

0.0s

- The **LabelEncoder** converts country names into numbers.

Split dataset into training and testing sets

```python
X_train, X_test, y_train, y_test = train_test_split(X_text, y_encoded, test_size=0.2, random_state=42)
```

0.0s Python

Using TensorFlow's TextVectorization layer for text processing

```python
vectorizer = tf.keras.layers.TextVectorization(max_tokens=10000, output_mode='int')
vectorizer.adapt(X_train)
```

0.0s Python

- Splits data into 80% training and 20% testing.
-
- Converts text into numerical vectors.
- max_tokens=10000,Uses a vocabulary size of 10,000 words.
- output_sequence_length=20 ,Truncates or pads sequences to 20 words.

Build Deep Learning Model

```
model = tf.keras.Sequential([
    vectorizer,
    tf.keras.layers.Embedding(input_dim=10000, output_dim=64, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(encoder.classes_), activation='softmax')  # Output layer
])
```

[70] ✓ 0.0s

- Embedding Layer: Converts word indices into dense 64-dimensional vectors.
- Bidirectional LSTM Layers: Capture text dependencies in both directions.
- Dense Layer: Uses 64 neurons with ReLU activation.
- Output Layer: Uses Softmax for multi-class classification.

```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=['accuracy']
)
```

- Uses **SparseCategoricalCrossentropy** (since target labels are integers).
- Optimized using **Adam**.
- Evaluates **accuracy**.

```python
# Train Model
history = model.fit(
    X_train, y_train,
    epochs=10,
    validation_data=(X_test, y_test)
)

# Plot Accuracy & Loss
history_dict = history.history

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(history_dict['accuracy'])
plt.plot(history_dict['val_accuracy'])
plt.title('Training & Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Accuracy', 'Validation Accuracy'])

plt.subplot(1, 2, 2)
plt.plot(history_dict['loss'])
plt.plot(history_dict['val_loss'])
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Loss', 'Validation Loss'])
```

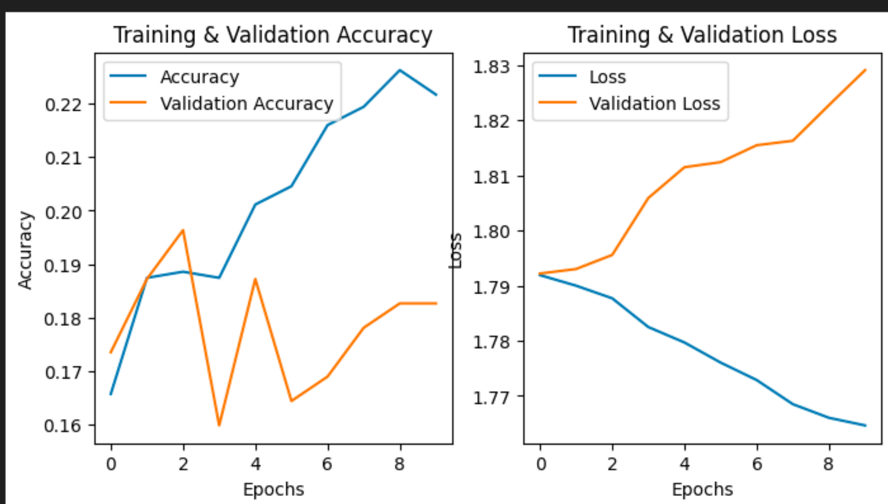
- Trains the model for **10 epochs**.
- Uses **validation data** to monitor progress.

```python
plt.show()

# Function to Predict Country
def predict_country(product_name):
    product_vector = vectorizer([product_name])
    prediction = model.predict(product_vector)
    predicted_class = np.argmax(prediction)
    return encoder.inverse_transform([predicted_class])[0]

# Example Prediction
sample_product = "Peanut Butter Cubes"
print(f"Predicted Country for '{sample_product}': {predict_country(sample_product)}")
```

```
Epoch 1/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 22s 197ms/step - accuracy: 0.1650 - loss: 1.7920 - val_accuracy: 0.1735 - val_loss: 1
Epoch 2/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 128ms/step - accuracy: 0.1903 - loss: 1.7898 - val_accuracy: 0.1872 - val_loss: 1.
Epoch 3/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 8s 221ms/step - accuracy: 0.1821 - loss: 1.7881 - val_accuracy: 0.1963 - val_loss: 1.
Epoch 4/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 90ms/step - accuracy: 0.1873 - loss: 1.7851 - val_accuracy: 0.1598 - val_loss: 1.8
Epoch 5/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 122ms/step - accuracy: 0.2110 - loss: 1.7772 - val_accuracy: 0.1872 - val_loss: 1.
Epoch 6/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 121ms/step - accuracy: 0.2010 - loss: 1.7702 - val_accuracy: 0.1644 - val_loss: 1.
Epoch 7/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 5s 93ms/step - accuracy: 0.2122 - loss: 1.7737 - val_accuracy: 0.1689 - val_loss: 1.8
Epoch 8/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 136ms/step - accuracy: 0.2260 - loss: 1.7637 - val_accuracy: 0.1781 - val_loss: 1.
Epoch 9/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 5s 129ms/step - accuracy: 0.2299 - loss: 1.7686 - val_accuracy: 0.1826 - val_loss: 1.
Epoch 10/10
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 153ms/step - accuracy: 0.2375 - loss: 1.7585 - val_accuracy: 0.1826 - val_loss: 1.
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step
Predicted Country for 'Peanut Butter Cubes': New Zealand
```

- Plots accuracy and loss over time to check performance.
- Converts product name to a vector.
- Predicts the most likely country using argmax().
- Predicts the **country** for **"Peanut Butter Cubes"**.

TEXT CLASSIFICATION USING CNN

This program predicts the **Country** based on the **Product** name using a **Convolutional Neural Network (CNN).**

Text clasification using CNN

```python
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

✓ 0.0s Python

- **Pandas & NumPy** → Handle dataset
- **Tokenizer & Pad Sequences** → Convert text into numerical sequences
- **Sequential Model** → Build CNN
- **Conv1D & GlobalMaxPooling1D** → Extract important text patterns
- **Dense Layers** → Perform classification
- **Scikit-learn** → Encode labels & evaluate model

Display dataset columns

```
print("Available Columns in Dataset:", data.columns)

text_column = "Product"
label_column = "Country"

if text_column not in data.columns or label_column not in data.columns:
    raise ValueError(f"Columns '{text_column}' or '{label_column}' not found in dataset!")
```

[80] ✓ 0.0s

```
Available Columns in Dataset: Index(['Sales Person', 'Country', 'Product', 'Date', 'Amount',
       'Boxes Shipped'],
      dtype='object')
```

Drop any missing values

```
data = data[[text_column, label_column]].dropna()
```

[81] ✓ 0.1s

Convert categorical labels to numerical values

```
label_encoder = LabelEncoder()
data[label_column] = label_encoder.fit_transform(data[label_column])
```

[82] ✓ 0.0s

- Loads CSV file
- Ensures required columns exist
- Removes missing values
- Encodes Country names into numbers

Tokenize text data

```
max_words = 5000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(data[text_column])
sequences = tokenizer.texts_to_sequences(data[text_column])
```

[83] ✓ 0.1s

Pad sequences

```
max_length = 100
X = pad_sequences(sequences, maxlen=max_length)
y = data[label_column]
```

[84] ✓ 0.0s

- **Tokenization:** Converts words into numerical sequences
- **Padding:** Ensures all inputs have the same length

Split dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

✓ 0.0s

Build CNN Model

```
model = Sequential([
    Embedding(input_dim=max_words, output_dim=50, input_length=max_length),
    Conv1D(250, 3, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(250, activation='relu'),
    Dense(len(data[label_column].unique()), activation='softmax')
])
```

- **Splits data into 75% training & 25% testing**

- Model Layers
- **Embedding Layer:** Converts words into vector representations
- **Conv1D Layer:** Identifies patterns in text
- **GlobalMaxPooling1D:** Extracts most important features

- **Dense Layers:** Fully connected layers for classification

Train Model

```
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))
```

✓ 12.5s

```
Epoch 1/5
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 79ms/step - accuracy: 0.1580 - loss: 1.7934 - val_accuracy: 0.1861 - val_loss: 1.
Epoch 2/5
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.1944 - loss: 1.7881 - val_accuracy: 0.1752 - val_loss: 1.
Epoch 3/5
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.1920 - loss: 1.7830 - val_accuracy: 0.1642 - val_loss: 1.
Epoch 4/5
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.2223 - loss: 1.7756 - val_accuracy: 0.1788 - val_loss: 1.
Epoch 5/5
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.2366 - loss: 1.7573 - val_accuracy: 0.1715 - val_loss: 1.
```

Predict

```
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)
```

[89] ✓ 0.7s

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step
```

- Loss Function: sparse_categorical_crossentropy for multi-class classification
- Optimizer: adam for efficient training
- Epochs: 5 (Train model for 5 iterations)
- Batch Size: 32 (Processes 32 samples at a time)

```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

[90] ✓ 0.0s

```
Accuracy: 0.17
Precision: 0.14
Recall: 0.17
F1-score: 0.13
```

- **Accuracy:** Overall correctness of the model
- **Precision:** Correct predictions out of all predicted positives
- **Recall:** Correct predictions out of all actual positives
- **F1-score:** Balance between precision & recall.