```
!pip install transformers pandas scikit-learn
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionM
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transform
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (
```

```
# Load dataset
df = pd.read_csv("340dataset.csv")
```

```
# Filter samples into short (<50 words) and long (>200 words) length categories
df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))
short_texts = df[df['word_count'] < 50]
long_texts = df[df['word_count'] > 200]

print(f"Total samples: {len(df)}")
print(f"Short text samples: {len(short_texts)}")
print(f"Long text samples: {len(long_texts)}")
```

```
Total samples: 100
Short text samples: 41
Long text samples: 46
```

```
# Splitting the dataset

# 70% for training, 15% for validation, 15% for testing
train_df, temp_df = train_test_split(df, test_size=0.30, random_state=42)  # 70% train, 30% temp
val_df, test_df = train_test_split(temp_df, test_size=0.50, random_state=42)  # Split temp into 50% val and 50% test

print(f"Training set size: {len(train_df)}")
print(f"Validation set size: {len(val_df)}")
print(f"Testing set size: {len(test_df)}")
```

```
Training set size: 70
Validation set size: 15
Testing set size: 15
```

```
# Loading tokenizer and model
model_name = "roberta-base-openai-detector"  # Publicly accessible model for AI detection
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

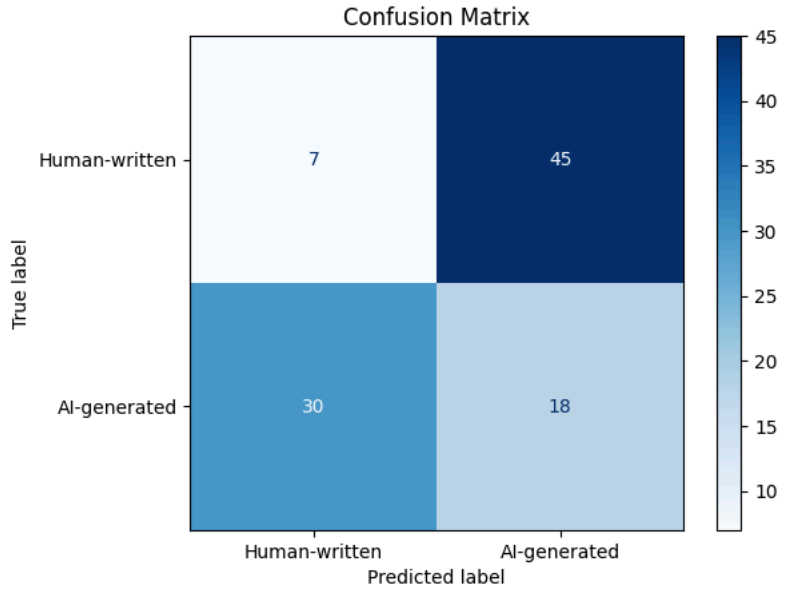
```
Some weights of the model checkpoint at roberta-base-openai-detector were not used when initializing RobertaForSequenceClass
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model trained on anothe
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model that you expe
```

```
def detect_ai_generated(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
        probs = torch.softmax(outputs.logits, dim=1)
        ai_prob = probs[0][1].item()  # Probability of being AI-generated
        return "AI-generated" if ai_prob > 0.5 else "Human-written", ai_prob



# Short text results
short_text_results = []
for text in short_texts['text']:
    prediction, prob = detect_ai_generated(text)
    short_text_results.append({
        'text': text,
        'label': 'Short',
        'prediction': prediction,
        'probability': prob
    })

# Long text results
long_text_results = []
for text in long_texts['text']:
    prediction, prob = detect_ai_generated(text)
    long_text_results.append({
        'text': text,
        'label': 'Long',
        'prediction': prediction,
        'probability': prob
    })

# Convert results to DataFrames
short_df = pd.DataFrame(short_text_results)
long_df = pd.DataFrame(long_text_results)

df['true_label'] = df['label'].apply(lambda x: 1 if x == 'AI-generated' else 0)


# Accuracy and AUROC for short texts
short_df['true_label'] = short_texts['true_label'].values
short_df['predicted_label'] = short_df['prediction'].apply(lambda x: 1 if x == 'AI-generated' else 0)
short_accuracy = accuracy_score(short_df['true_label'], short_df['predicted_label'])
short_auroc = roc_auc_score(short_df['true_label'], short_df['probability'])

# Accuracy and AUROC for long texts
long_df['true_label'] = long_texts['true_label'].values
long_df['predicted_label'] = long_df['prediction'].apply(lambda x: 1 if x == 'AI-generated' else 0)
long_accuracy = accuracy_score(long_df['true_label'], long_df['predicted_label'])
long_auroc = roc_auc_score(long_df['true_label'], long_df['probability'])

print(f"Short Text Accuracy: {short_accuracy:.2f}, AUROC: {short_auroc:.2f}")
print(f"Long Text Accuracy: {long_accuracy:.2f}, AUROC: {long_auroc:.2f}")
```

```
Short Text Accuracy: 0.37, AUROC: 0.33
Long Text Accuracy: 0.15, AUROC: 0.09
```

```
def calculate_metrics_with_confusion_matrix(df):
    results = []
    for text in df['text']:
        prediction, prob = detect_ai_generated(text)
        results.append({
            'text': text,
            'true_label': df[df['text'] == text]['true_label'].values[0],
            'predicted_label': 1 if prediction == "AI-generated" else 0,
            'probability': prob,
        })

    results_df = pd.DataFrame(results)

    precision = precision_score(results_df['true_label'], results_df['predicted_label'])
    recall = recall_score(results_df['true_label'], results_df['predicted_label'])
    f1 = f1_score(results_df['true_label'], results_df['predicted_label'])

    # Confusion matrix
    cm = confusion_matrix(results_df['true_label'], results_df['predicted_label'])
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Human-written", "AI-generated"])
```

```
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Human-written", "AI-generated"])
        disp.plot(cmap=plt.cm.Blues)
        plt.title("Confusion Matrix")
        plt.show()

        return precision, recall, f1

precision, recall, f1 = calculate_metrics_with_confusion_matrix(df)

# Output metrics
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```




```
Precision: 0.2857142857142857
Recall: 0.375
F1 Score: 0.32432432432432434
```