American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Natural Language Processing

Final-Term Project Report

Fall 2025-2026

Section: B

Group:08

Project Contributions

SL #	Student Name & ID	Contributions (mention every part of the project where you contributed)
1.	Name: MD. MARUF HASAN CHOWDHURY ID:22-48582-3	Code(preprocessing & model_2 implementation) Report- implementation, Result
2.	Name: DIBAJIT ROY ID: 22-48569-3	Code(model_1 implemention) Report: methodology
3.	Name: MD. OSMAN GANI ID:22-48535-3	Dataset collection Report: Beckground
4.	Name: A.K.M SHAHRIAR EMTIAZ ID: 22-48012-2	Report: Abstract, introduction, conclusion

AI Content Detection Using BERT and DistilBERT

MD. MARUF HASAN CHOWDHURY, DIBAJIT ROY, MD. OSMAN GANI, and A.K.M SHAHRIAR EMTIAZ
American International University-Bangladesh Dhaka, Bangladesh
{22-48582-3@student.aiub.edu, 22-48569-3@student.aiub.edu, 22-48533@student.aiub.edu, 22-48012-2@student.aiub.edu}

Abstract

This project uses two transformer-based language models, BERT and DistilBERT, to detect AI-generated text. The main objective of this work is to evaluate how effective these models are in distinguishing human-written text from AI-generated text. This problem is very important in natural language processing from the perspective of credibility and academic integrity of online content. For this purpose, both models are fine-tuned and evaluated using a public dataset including human-written and AI-generated texts. 10,000 balanced samples are selected from the large dataset for fair comparison. The same preprocessing steps and training configuration are applied to both models, so that the comparison is accurate and unbiased. The experimental results show that both BERT and DistilBERT provide good performance in recognizing AI-generated text. However, DistilBERT achieves results close to BERT with fewer parameters and less computational cost. This proves that it is possible to effectively recognize AI content even using lightweight and efficient transformer models.

Keywords-AI content detection, BERT, DistilBERT, transformer model

1 Introduction

Currently, with the rapid development of artificial intelligence, AI-generated texts are being easily generated. These AI-generated texts are now widely used in academic writing, online articles, social media posts, and various digital platforms. Although this technology is useful in many areas, it also poses serious problems such as content credibility, academic integrity, and the spread of misinformation. Therefore, distinguishing human-written texts from AI-generated texts has become an important research problem in Natural Language Processing (NLP). Conventional text classification or rule-based methods usually rely on simple linguistic features, which cannot accurately capture the complex structure and contextual features of modern AI-generated texts. To overcome this limitation, transformer-based language models are currently widely used. BERT is a powerful transformer model that is able to understand the deep meaning and context of texts by using two-way attention. However, the BERT model is relatively large and requires more computational resources. To solve this problem, a lightweight version called DistilBERT has been developed, which tries to provide performance close to BERT while using fewer parameters. In this project, two models BERT and DistilBERT have been used to recognize human-written and AI-generated text. The main objective of this work is to analyse how effective a computationally efficient lightweight transformer model can be in AI-generated content detection. This project compares the performance of BERT and DistilBERT by fine-tuning and evaluating the two models on the same dataset and in the same experimental environment.

2 Background

This project uses two transformer-based language models BERT and DistilBERT to recognize AI-generated and human-written text. Transformer architecture is very effective in understanding the context of natural language, as it can analyse the relationship between each word in a sentence and other words using a self-attention mechanism. This section briefly explains the architecture and working principle of BERT and DistilBERT models.

2.1 BERT Architecture

BERT (Bidirectional Encoder Representations from Transformers) is a deep bidirectional transformer encoder-based language model. It can simultaneously consider the context on the left and right sides of a sentence, which makes it more powerful than conventional unidirectional language models. In the BERT model, the input text is first divided into tokens using the WordPiece tokenizer. Each input sequence is preceded by a special [CLS] token and used for sentence segmentation using [SEP] tokens. The output vector of [CLS] tokens is used for the classification task. The input representation is formed as the sum of three embeddings :token embedding, segment embedding, and position embedding. BERT mainly works in two stages: pre-training and fine-tuning. In the pre-training stage, it learns a deep contextual representation of the language using the Masked Language Model (MLM) and Next Sentence Prediction (NSP). In the fine-tuning stage, the pre-trained BERT model is trained by adding a classification layer for a specific task (in this case, AI content detection).

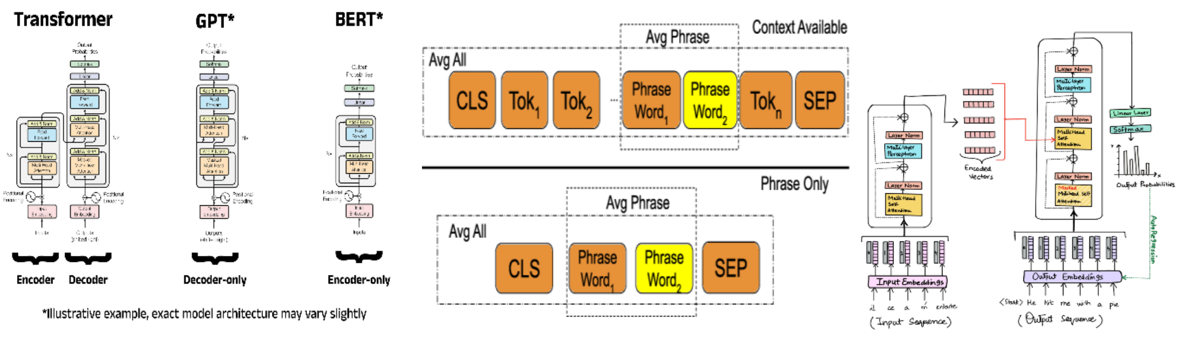


Figure 1: Architecture of BERT model showing input representation and transformer encoding layers

2.2 DistilBERT Architecture

DistilBERT is a lightweight version of BERT, which is built using knowledge distillation techniques. In this method, a larger model (teacher: BERT) is trained on a smaller model (student: DistilBERT). As a result, DistilBERT is able to provide almost the same performance with fewer parameters. Although DistilBERT uses the same transformer encoder architecture as BERT, it has fewer transformer layers, and the Next Sentence Prediction (NSP) task has been eliminated. As a result, the model trains faster and uses fewer computational resources. However, it uses a Masked Language Model (MLM) to learn contextual information about the language. In this project, the DistilBERT model has been fine-tuned following the same tokenization, preprocessing, and training pipeline as BERT. Experimental results show that despite having fewer parameters, DistilBERT is able to provide performance close to or slightly better than BERT, which highlights it as a viable alternative for real-world applications.

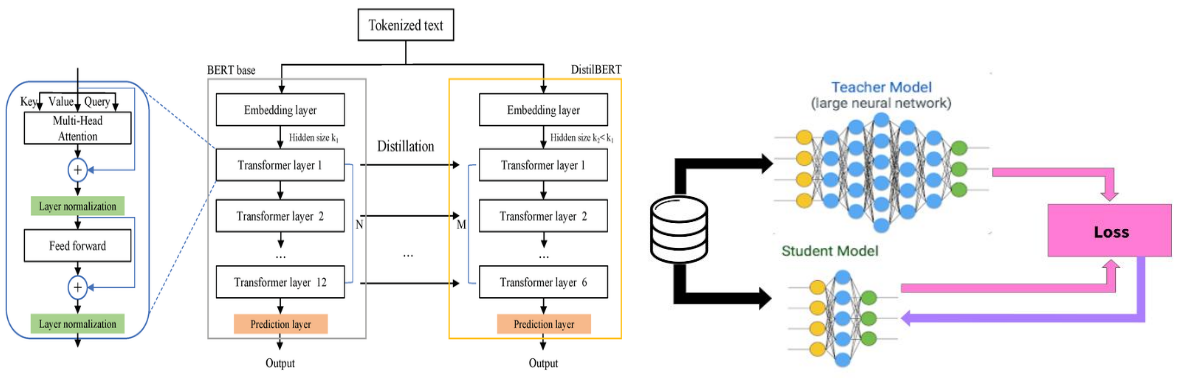


Figure 2: Knowledge distillation process from BERT to DistilBERT

3 Methodology

This section presents the complete methodology followed in the AI-generated content detection project step by step. The entire work was completed following a specific workflow, which includes every step from data collection to model evaluation. The above figure shows the overall methodological steps of this project.

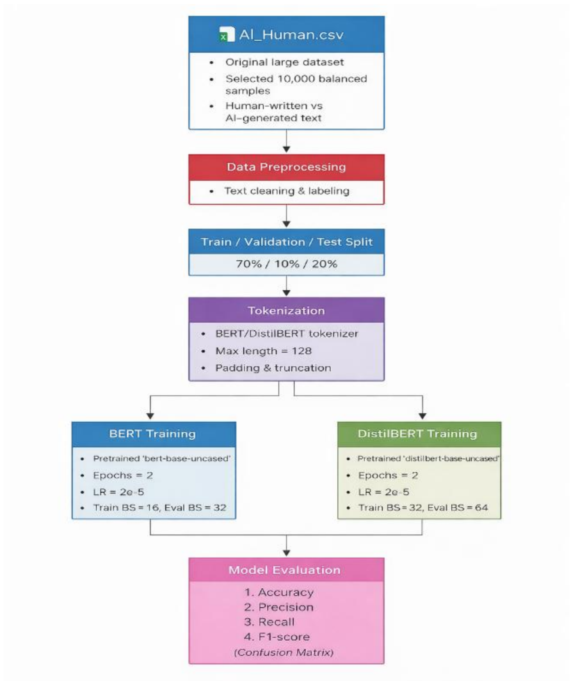


Figure: Workflow of AI-generated content detection using BERT and DistilBERT

3.1Data Collection

This project uses a public dataset to identify AI-generated and human-written text. The dataset is taken from Kaggle and includes both human-written and artificial intelligence-generated text. Due to the large size of the original dataset, it was not practical to use the entire data directly. Therefore, a total of 10,000 balanced samples were selected from the original dataset in this study. The selected dataset contains equal numbers of human-written and AI-generated text, so that class imbalances do not occur during model training and evaluation and the results are reliable.

3.2 Data Preprocessing

After data collection, various preprocessing steps were followed to make the text data suitable for model training. First, blank or incomplete text data was removed. Then, corresponding labels were assigned to each text, where human-written and AI-generated text were divided into separate classes. As part of the preprocessing step, the dataset was divided into training, validation, and test sets. This division was done in the ratio of 70% training, 10% validation, and 20% test. Then, the texts were tokenized using a tokenizer suitable for BERT and DistilBERT models. Padding and truncation were applied during tokenization, and the maximum sequence length was set to 128.

3.3 Model Training

This study uses two pretrained transformer-based models BERT and DistilBERT for the sequence classification problem. Both models are fine-tuned for the training process. Tokenized training data is used for the training process, and the model performance is monitored at the end of each epoch using validation data. For fair comparison, the same dataset and the same training strategy are followed for both BERT and DistilBERT models. The Hugging Face Trainer API is used to complete the training process, which makes the model training and evaluation process simple and controllable.

3.4 Model Evaluation

After the model training was completed, the performance of both models was evaluated using the test dataset. Accuracy, Precision, Recall and F1-score were used for this evaluation. In addition, the ability to correctly recognize human-written and AI-generated text was analysed by creating a confusion matrix. The results obtained through this evaluation step are presented and compared in detail in the next Result Analysis section, where the performance of the BERT and DistilBERT models is analysed.

4 Implementation

This section presents detailed information about the implementation of the AI-generated content detection system. The programming environment, libraries, data processing methods used, and the implementation steps of the BERT and DistilBERT models are described here. The entire implementation was done in the Google Colab environment using the Python programming language and the pretrained transformer models were fine-tuned using the Hugging Face Transformers library.

Programming Environment and Libraries

The main tools and libraries used to implement the project are presented in the table 4.1 below.

Table 4.1: Programming Environment and Libraries

Programming Language	Python
Development Environment	Google Colab
Deep Learning Framework	PyTorch
Transformer Library	Hugging Face Transformers
Evaluation Library	Scikit-learn
Hardware Support	T4-GPU

4.1 BERT Implementation

The BERT model is implemented using the pretrained bert-base-uncased model from Hugging Face. This model is initialized with num_labels=2 for binary text classification. The corresponding tokenizer is used to process the input text and padding, and truncation are applied during tokenization, where the maximum sequence length is set to 128. After tokenization is completed, only the necessary columns input_ids, attention_mask, and labels are kept from the dataset and the remaining columns are omitted. The dataset is then converted to PyTorch format so that it can be easily trained using the Trainer API. The Trainer API of Hugging Face is used to manage the training and validation process of the BERT model. Training Arguments is used to define the training configuration, where learning rate, batch size, number of epochs, and weight decay are specified. A custom metric function is used to evaluate model performance, through which Accuracy, Precision, Recall, and F1-score are calculated.

The training configuration of the BERT model is summarized in the table 4.2 below.

Table 4.2: BERT Training Configuration

Parameter	**Value**
Pretrained Model	bert-base-uncased
Task Type	Binary Classification
Number of Labels	2
Max Sequence Length	128
Learning Rate	2e-5
Train Batch Size	16
Evaluation Batch Size	32
Number of Epochs	2
Weight Decay	0.01
Training API	Hugging Face Trainer

4.2 DistilBERT Implementation

The pretrained Distilbert-base-uncased model of Hugging Face is used for the DistilBERT implementation. DistilBERT is a lightweight version of BERT, which provides fast training and inference using fewer parameters. This model is also initialized with num_labels =2 for binary classification. The tokenization, dataset formatting, and training pipeline for DistilBERT are kept the same as BERT, so that the comparison between the two models is fair. The DistilBERT model is fine-tuned using the same tokenizer strategy, the same sequence length, and the same evaluation metrics. The Trainer API of Hugging Face is also used for training and evaluation.

The training configuration of the DistilBERT model is presented in the table 4.3 below.

Table 4.3: DistilBERT Training Configuration

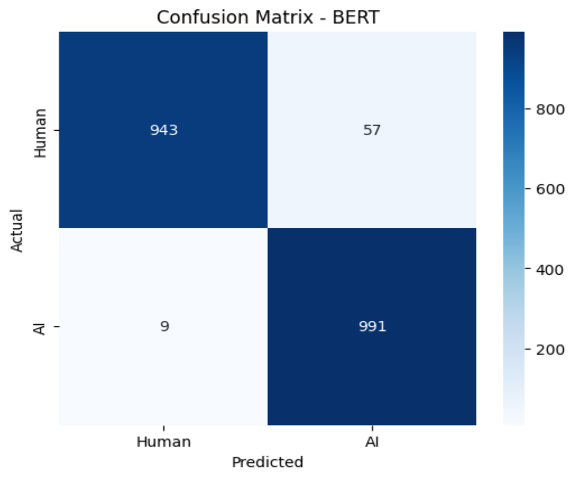
Parameter	**Value**
Pretrained Model	distilbert-base-uncased
Task Type	Binary Classification
Number of Labels	2
Max Sequence Length	128
Learning Rate	2e-5
Train Batch Size	32
Evaluation Batch Size	64
Number of Epochs	2
Weight Decay	0.01
Training API	Hugging Face Trainer

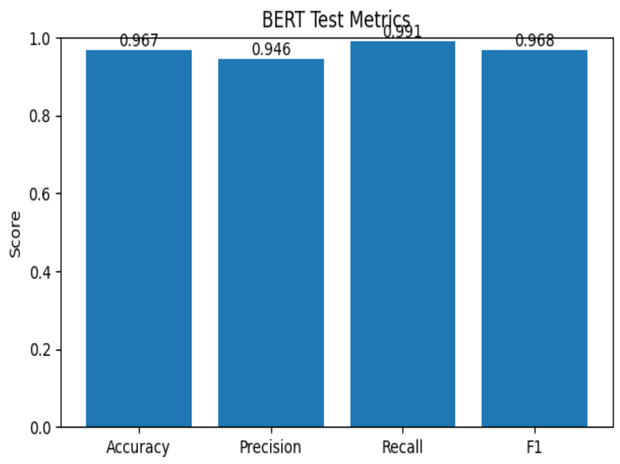
5 Result Analysis

This section analyses the performance of BERT and DistilBERT models used for AI-generated content detection. Both models are evaluated on the same dataset and in the same experimental environment, so that the comparison of results is fair and reliable. Accuracy, Precision, Recall, and F1-score are used to evaluate the performance of the models.

BERT:

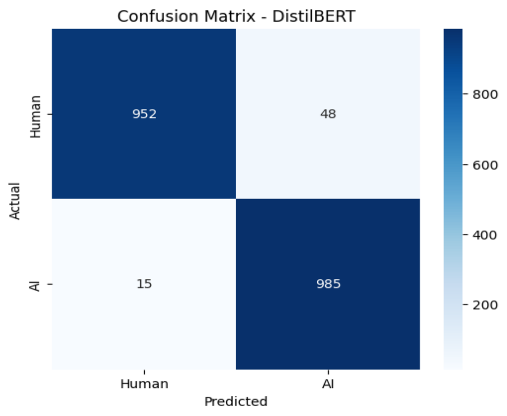
Analysing the results of the BERT model shows that it provides strong performance in distinguishing between human-written and AI-generated text. According to the confusion matrix, most of the text is correctly classified and the misclassification rate is relatively low. Test metrics also show that BERT achieves high Accuracy and F1-score, indicating its effective contextual understanding capabilities.

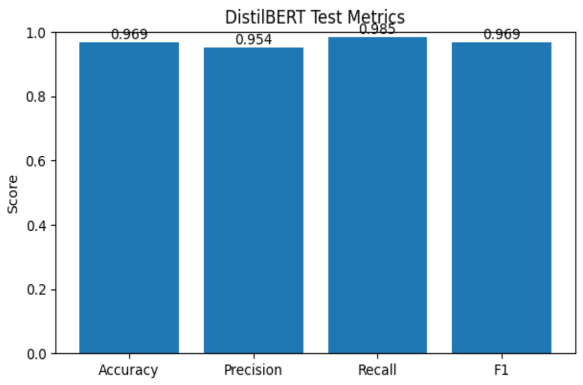




DistilBERT:

Analysing the results of the DistilBERT model, it is seen that despite being a lightweight model, it played an effective role in recognizing human-written and AI-generated text. The confusion matrix shows that DistilBERT was able to recognize both classes of text accurately. According to the test metrics, its Accuracy, Precision, Recall and F1-score provided values close to BERT.





Analysing the comparative results of BERT and DistilBERT, it is seen that DistilBERT achieves almost equivalent performance to BERT, using fewer parameters and less computational cost. Although BERT provides slightly better results in some cases, the efficiency and fast processing power of DistilBERT make it a viable alternative.

Comparative results of BERT and DistilBERT:

Model	Accuracy	Precision	Recall	F1
BERT	0.967	0.946	0.991	0.968
DistilBERT	0.969	0.954	0.985	0.969

Overall, these experimental results demonstrate that transformer-based models are highly effective for AI-generated content detection. In particular, DistilBERT can be a useful solution for real-world applications, as it provides effective results while using fewer resources.

6 Conclusion

In this project, two transformer-based models, BERT and DistilBERT, have been successfully implemented and evaluated for recognizing AI-generated and human-written text. Both models were trained with the same preprocessing steps, tokenization techniques, and training configurations using 10,000 balanced samples selected from a public dataset, to ensure fair comparison. Experimental results show that both BERT and DistilBERT models achieve high accuracy in recognizing AI-generated content. Although DistilBERT achieves slightly higher accuracy, the difference in performance between the two models is very small, indicating that their performance is almost equivalent. However, DistilBERT is able to achieve these results using fewer parameters and fewer computational resources, which makes it a more efficient and realistic model. The results of this research can be effectively used in important areas such as protecting the credibility of online content, maintaining academic integrity, and recognizing AI-generated information. There is scope to expand this work further in the future using larger datasets, different languages, and more advanced transformer-based models.

References

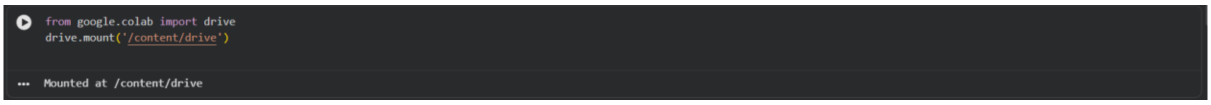
1. J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, “Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), pp. 4171–4186, 2019.[Online]. Available: https://arxiv.org/abs/1810.04805

2. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT: a distilled version of BERT: smaller, faster, cheaper and lighter, ”arXiv preprint arXiv:1910.01108, 2019.[Online]. Available: https://arxiv.org/abs/1910.01108

3. Syed Ali, “AI-generated vs Human-written Text Classification,” Kaggle, 2025.Available:https://www.kaggle.com/code/syedali110/ai-generated-vs-human-text-95-accuracy/input

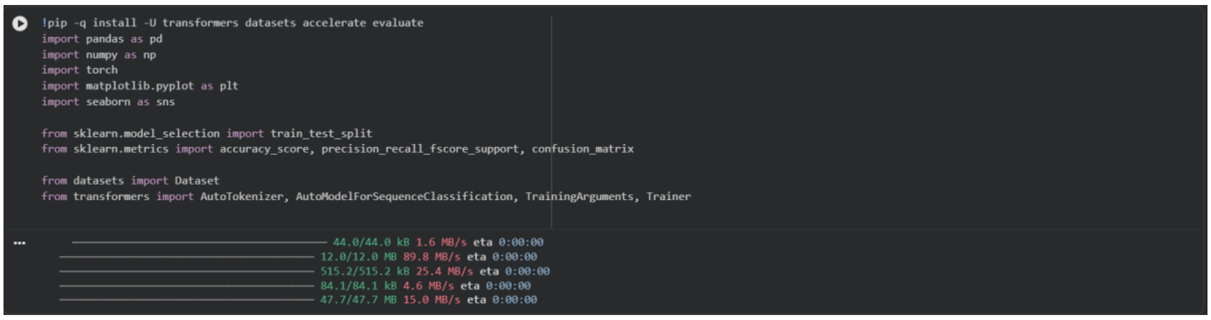
Project Code

Model_1: BERT



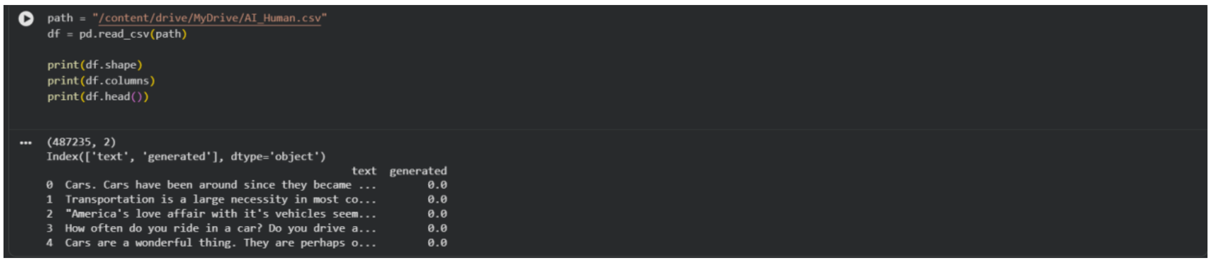
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```



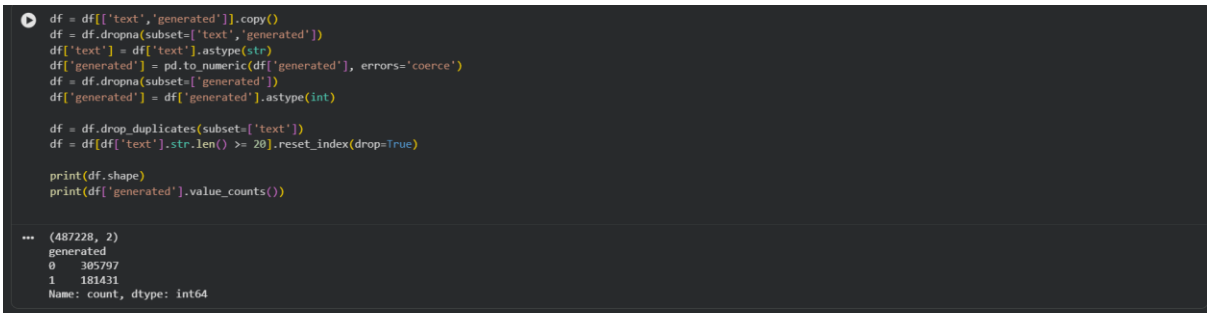
```
!pip -q install -U transformers datasets accelerate evaluate
import pandas as pd
import numpy as np
import torch
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix

from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
```

```
     ──────────────────────────────────────── 44.0/44.0 kB 1.6 MB/s eta 0:00:00
   ──────────────────────────────────────── 12.0/12.0 MB 89.8 MB/s eta 0:00:00
   ──────────────────────────────────────── 515.2/515.2 kB 25.4 MB/s eta 0:00:00
   ──────────────────────────────────────── 84.1/84.1 kB 4.6 MB/s eta 0:00:00
   ──────────────────────────────────────── 47.7/47.7 MB 15.0 MB/s eta 0:00:00
```



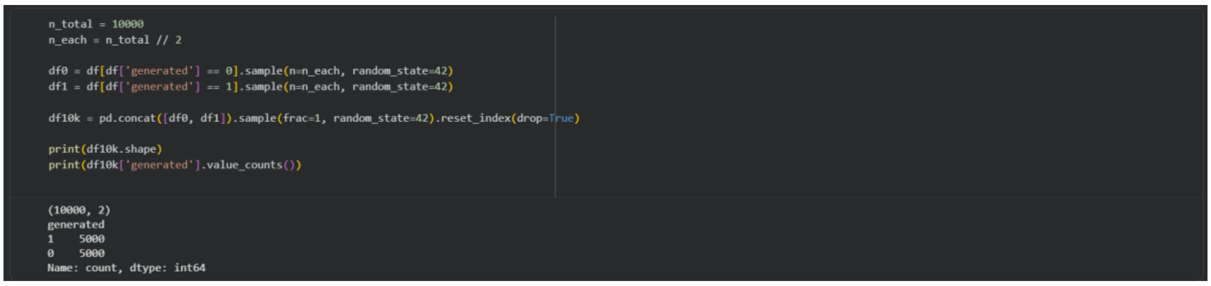
```
path = "/content/drive/MyDrive/AI_Human.csv"
df = pd.read_csv(path)

print(df.shape)
print(df.columns)
print(df.head())
```

```
(487235, 2)
Index(['text', 'generated'], dtype='object')
                                                text  generated
0  Cars. Cars have been around since they became ...        0.0
1  Transportation is a large necessity in most co...        0.0
2  "America's love affair with it's vehicles seem...        0.0
3  How often do you ride in a car? Do you drive a...        0.0
4  Cars are a wonderful thing. They are perhaps o...        0.0
```



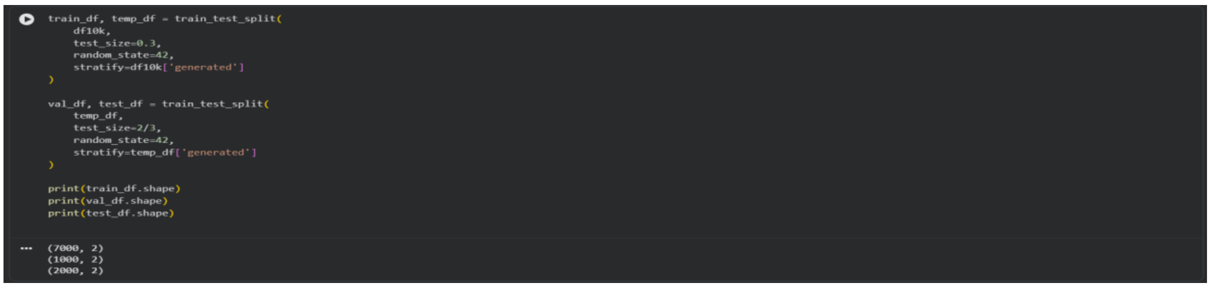
```
df = df[['text','generated']].copy()
df = df.dropna(subset=['text','generated'])
df['text'] = df['text'].astype(str)
df['generated'] = pd.to_numeric(df['generated'], errors='coerce')
df = df.dropna(subset=['generated'])
df['generated'] = df['generated'].astype(int)

df = df.drop_duplicates(subset=['text'])
df = df[df['text'].str.len() >= 20].reset_index(drop=True)

print(df.shape)
print(df['generated'].value_counts())
```

```
(487228, 2)
generated
0    305797
1    181431
Name: count, dtype: int64
```



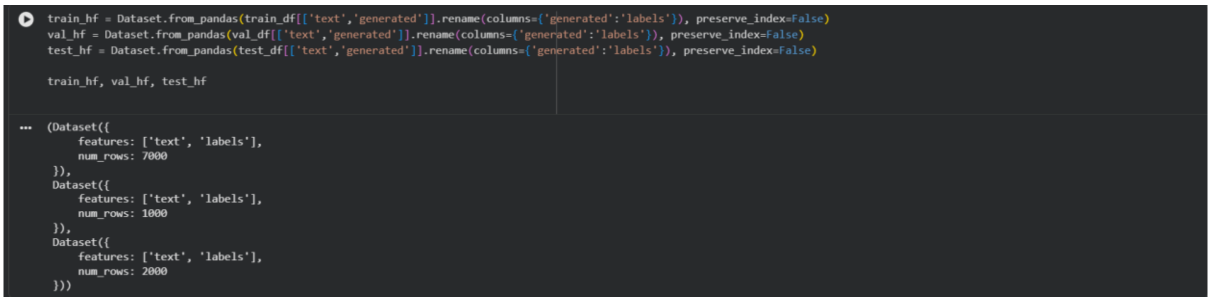
```
n_total = 10000
n_each = n_total // 2

df0 = df[df['generated'] == 0].sample(n=n_each, random_state=42)
df1 = df[df['generated'] == 1].sample(n=n_each, random_state=42)

df10k = pd.concat([df0, df1]).sample(frac=1, random_state=42).reset_index(drop=True)

print(df10k.shape)
print(df10k['generated'].value_counts())
```

```
(10000, 2)
generated
1    5000
0    5000
Name: count, dtype: int64
```



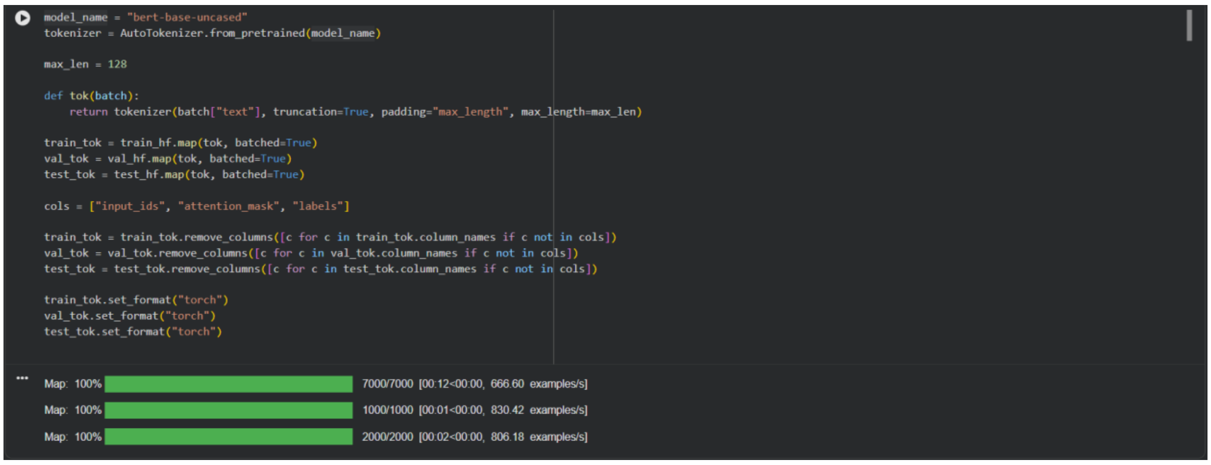
```
train_df, temp_df = train_test_split(
    df10k,
    test_size=0.3,
    random_state=42,
    stratify=df10k['generated']
)

val_df, test_df = train_test_split(
    temp_df,
    test_size=2/3,
    random_state=42,
    stratify=temp_df['generated']
)

print(train_df.shape)
print(val_df.shape)
print(test_df.shape)
```

```
(7000, 2)
(1000, 2)
(2000, 2)
```



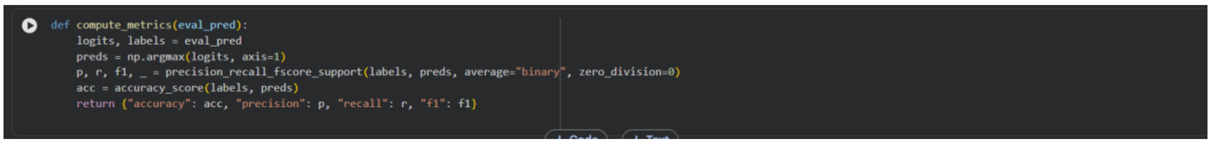
```
train_hf = Dataset.from_pandas(train_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)
val_hf = Dataset.from_pandas(val_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)
test_hf = Dataset.from_pandas(test_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)

train_hf, val_hf, test_hf
```

```
(Dataset({
     features: ['text', 'labels'],
     num_rows: 7000
 }),
 Dataset({
     features: ['text', 'labels'],
     num_rows: 1000
 }),
 Dataset({
     features: ['text', 'labels'],
     num_rows: 2000
 }))
```

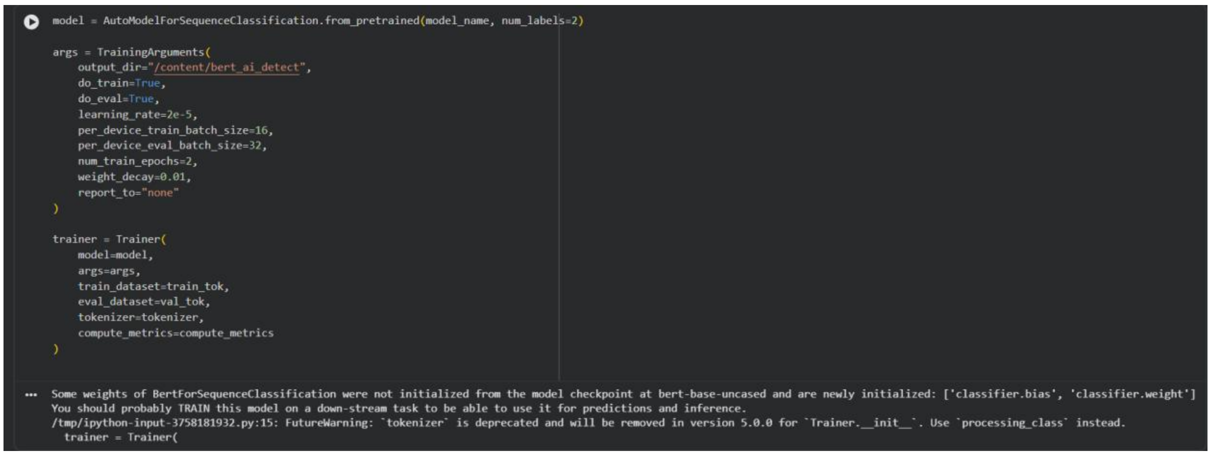


```
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

max_len = 128

def tok(batch):
    return tokenizer(batch["text"], truncation=True, padding="max_length", max_length=max_len)

train_tok = train_hf.map(tok, batched=True)
val_tok = val_hf.map(tok, batched=True)
test_tok = test_hf.map(tok, batched=True)

cols = ["input_ids", "attention_mask", "labels"]

train_tok = train_tok.remove_columns([c for c in train_tok.column_names if c not in cols])
val_tok = val_tok.remove_columns([c for c in val_tok.column_names if c not in cols])
test_tok = test_tok.remove_columns([c for c in test_tok.column_names if c not in cols])

train_tok.set_format("torch")
val_tok.set_format("torch")
test_tok.set_format("torch")
```

```
Map: 100%  7000/7000 [00:12<00:00, 666.60 examples/s]
Map: 100%  1000/1000 [00:01<00:00, 830.42 examples/s]
Map: 100%  2000/2000 [00:02<00:00, 806.18 examples/s]
```

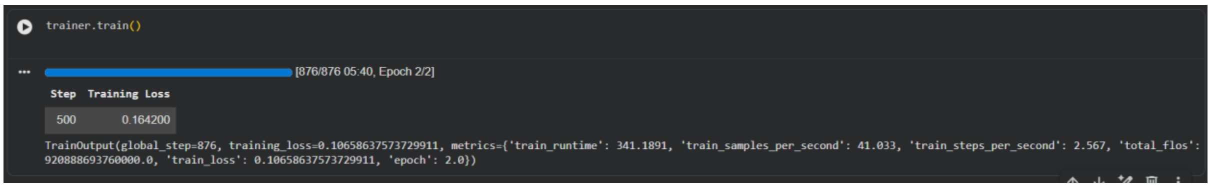


```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    p, r, f1, _ = precision_recall_fscore_support(labels, preds, average="binary", zero_division=0)
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "precision": p, "recall": r, "f1": f1}
```



```
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

args = TrainingArguments(
    output_dir="/content/bert_ai_detect",
    do_train=True,
    do_eval=True,
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=2,
    weight_decay=0.01,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_tok,
    eval_dataset=val_tok,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-3758181932.py:15: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

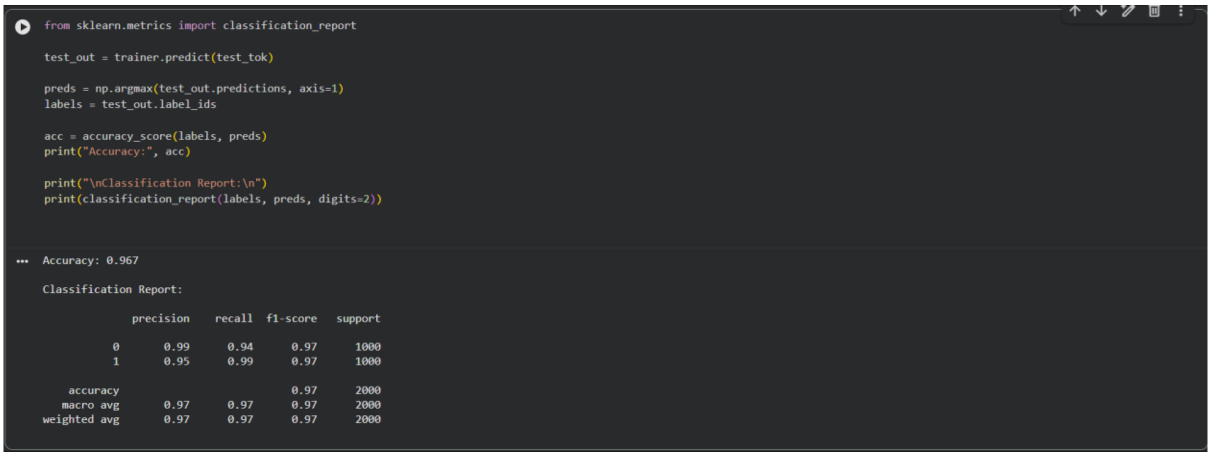


```
trainer.train()
```

```
[876/876 05:40, Epoch 2/2]
```

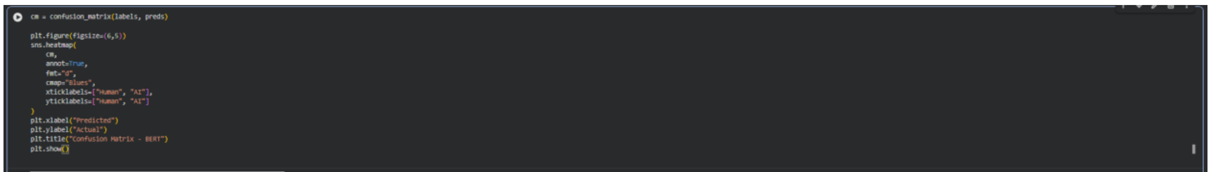
Step	Training Loss
500	0.164200

```
TrainOutput(global_step=876, training_loss=0.10658637573729911, metrics={'train_runtime': 341.1891, 'train_samples_per_second': 41.033, 'train_steps_per_second': 2.567, 'total_flos': 920888693760000.0, 'train_loss': 0.10658637573729911, 'epoch': 2.0})
```

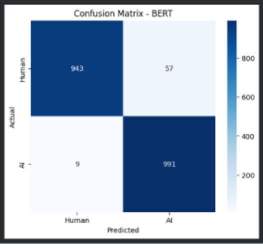


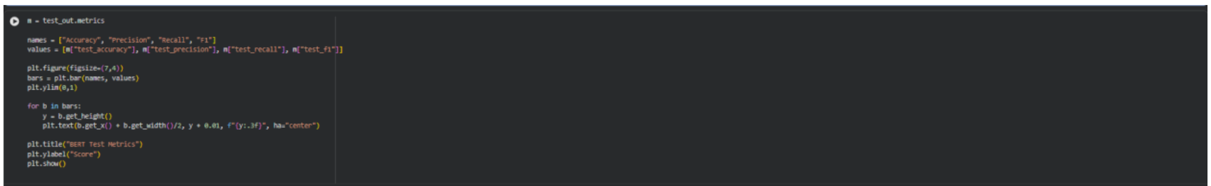
```
from sklearn.metrics import classification_report

test_out = trainer.predict(test_tok)

preds = np.argmax(test_out.predictions, axis=1)
labels = test_out.label_ids

acc = accuracy_score(labels, preds)
print("Accuracy:", acc)

print("\nClassification Report:\n")
print(classification_report(labels, preds, digits=2))
```

```
Accuracy: 0.967

Classification Report:

              precision    recall  f1-score   support

           0       0.99      0.94      0.97      1000
           1       0.95      0.99      0.97      1000

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000
```

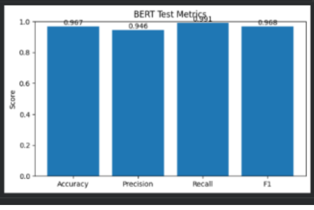


```
cm = confusion_matrix(labels, preds)

plt.figure(figsize=(6,5))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Human", "AI"],
    yticklabels=["Human", "AI"]
)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - BERT")
plt.show()
```

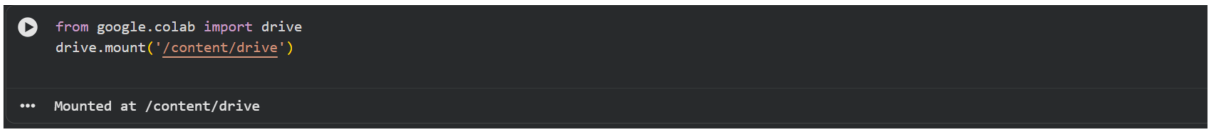




```
m = test_out.metrics

names = ["Accuracy", "Precision", "Recall", "F1"]
values = [m["test_accuracy"], m["test_precision"], m["test_recall"], m["test_f1"]]

plt.figure(figsize=(7,4))
bars = plt.bar(names, values)
plt.ylim(0,1)

for b in bars:
    y = b.get_height()
    plt.text(b.get_x() + b.get_width()/2, y + 0.01, f"{y:.3f}", ha="center")

plt.title("BERT Test Metrics")
plt.ylabel("Score")
plt.show()
```

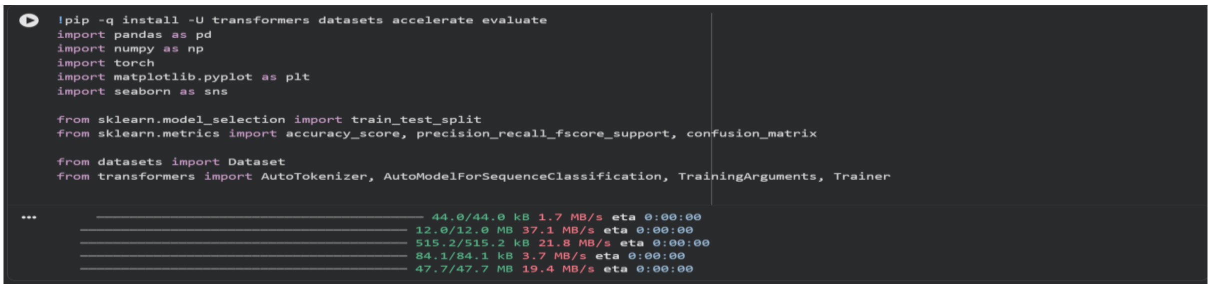


Model_2: DistilBERT



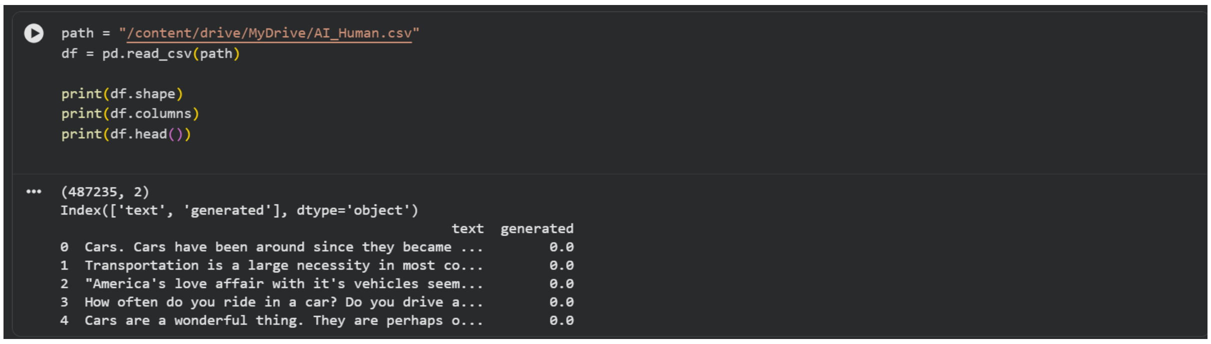
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```



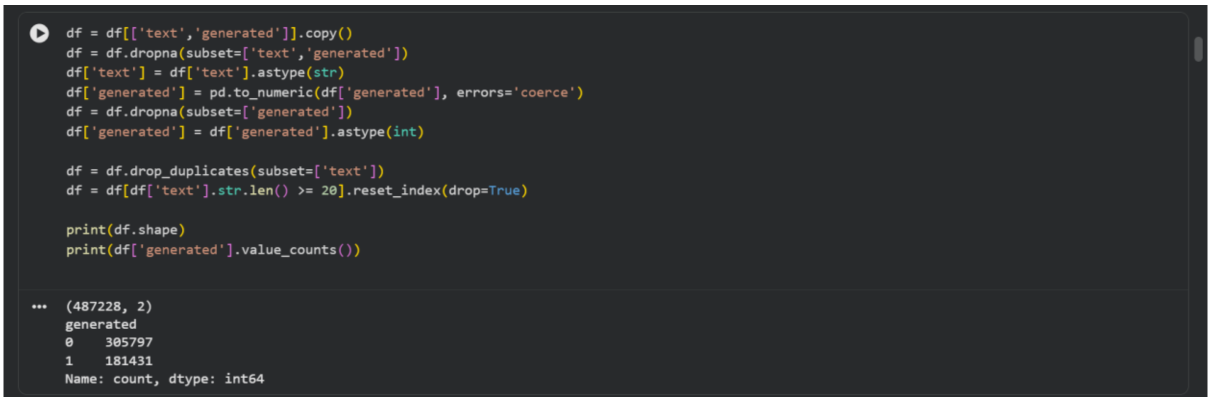
```
!pip -q install -U transformers datasets accelerate evaluate
import pandas as pd
import numpy as np
import torch
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix

from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.0/44.0 kB 1.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 37.1 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 515.2/515.2 kB 21.8 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 3.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.7/47.7 MB 19.4 MB/s eta 0:00:00
```



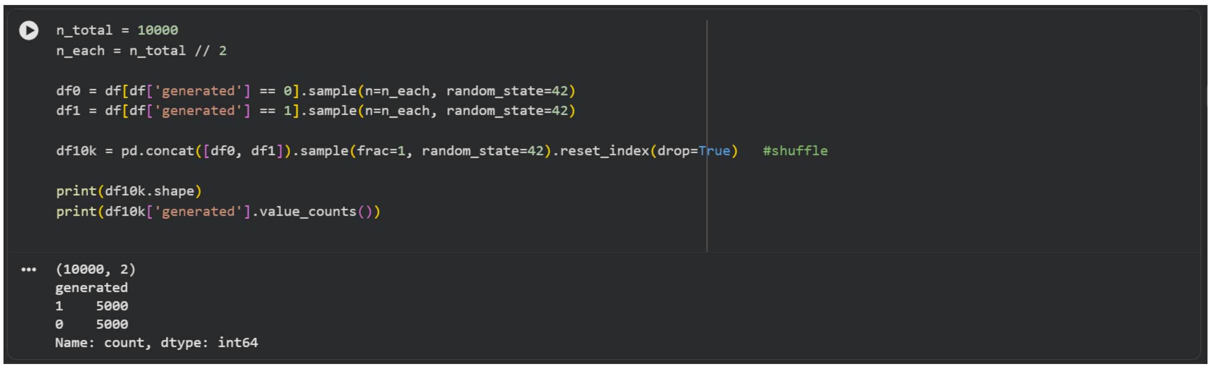
```
path = "/content/drive/MyDrive/AI_Human.csv"
df = pd.read_csv(path)

print(df.shape)
print(df.columns)
print(df.head())
```

```
(487235, 2)
Index(['text', 'generated'], dtype='object')
                                                text  generated
0  Cars. Cars have been around since they became ...        0.0
1  Transportation is a large necessity in most co...        0.0
2  "America's love affair with it's vehicles seem...        0.0
3  How often do you ride in a car? Do you drive a...        0.0
4  Cars are a wonderful thing. They are perhaps o...        0.0
```



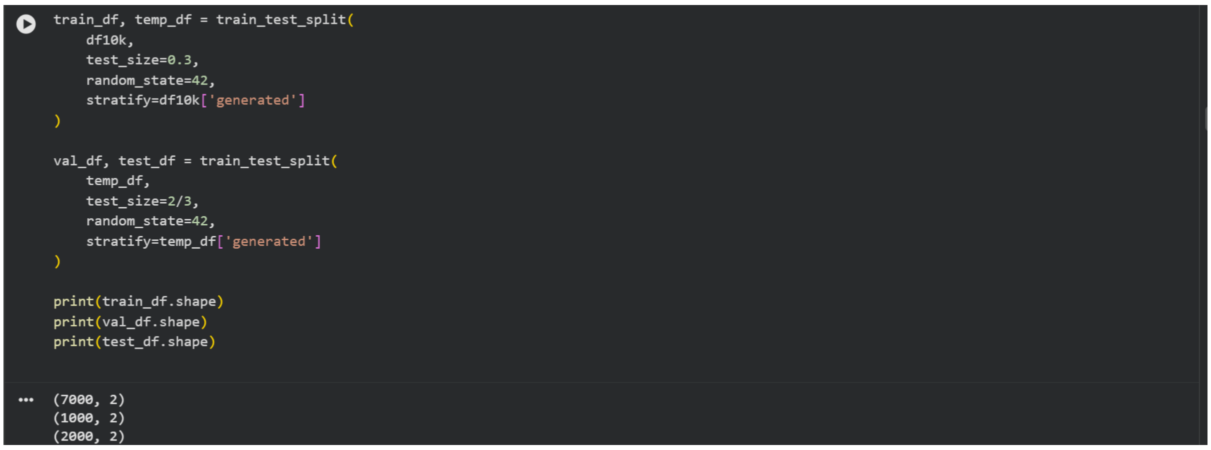
```
df = df[['text','generated']].copy()
df = df.dropna(subset=['text','generated'])
df['text'] = df['text'].astype(str)
df['generated'] = pd.to_numeric(df['generated'], errors='coerce')
df = df.dropna(subset=['generated'])
df['generated'] = df['generated'].astype(int)

df = df.drop_duplicates(subset=['text'])
df = df[df['text'].str.len() >= 20].reset_index(drop=True)

print(df.shape)
print(df['generated'].value_counts())
```

```
(487228, 2)
generated
0    305797
1    181431
Name: count, dtype: int64
```



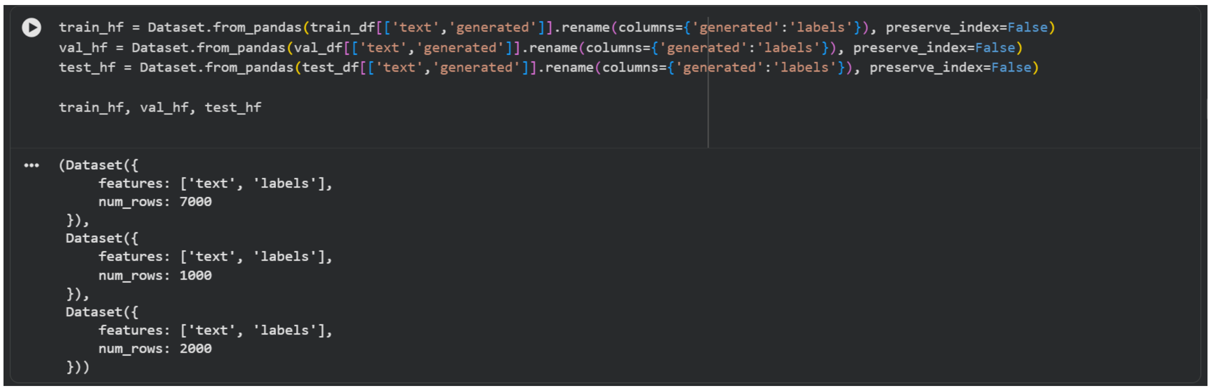
```
n_total = 10000
n_each = n_total // 2

df0 = df[df['generated'] == 0].sample(n=n_each, random_state=42)
df1 = df[df['generated'] == 1].sample(n=n_each, random_state=42)

df10k = pd.concat([df0, df1]).sample(frac=1, random_state=42).reset_index(drop=True)   #shuffle

print(df10k.shape)
print(df10k['generated'].value_counts())
```

```
(10000, 2)
generated
1    5000
0    5000
Name: count, dtype: int64
```



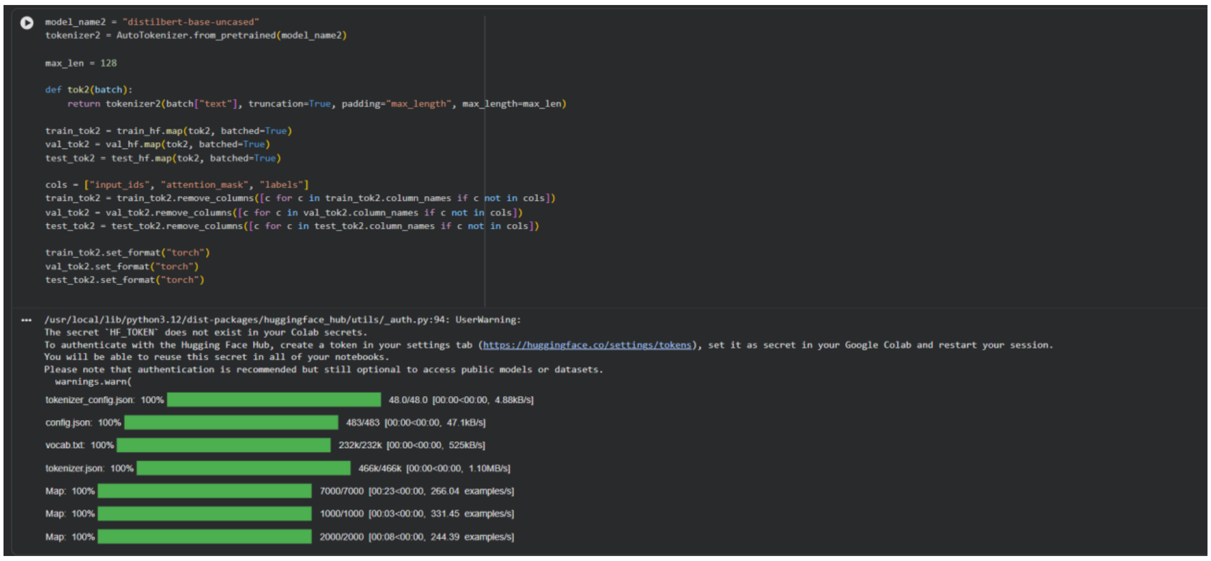
```
train_df, temp_df = train_test_split(
    df10k,
    test_size=0.3,
    random_state=42,
    stratify=df10k['generated']
)

val_df, test_df = train_test_split(
    temp_df,
    test_size=2/3,
    random_state=42,
    stratify=temp_df['generated']
)

print(train_df.shape)
print(val_df.shape)
print(test_df.shape)
```

```
(7000, 2)
(1000, 2)
(2000, 2)
```



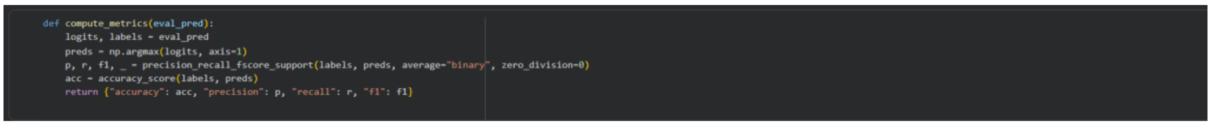
```
train_hf = Dataset.from_pandas(train_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)
val_hf = Dataset.from_pandas(val_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)
test_hf = Dataset.from_pandas(test_df[['text','generated']].rename(columns={'generated':'labels'}), preserve_index=False)

train_hf, val_hf, test_hf
```

```
(Dataset({
     features: ['text', 'labels'],
     num_rows: 7000
 }),
 Dataset({
     features: ['text', 'labels'],
     num_rows: 1000
 }),
 Dataset({
     features: ['text', 'labels'],
     num_rows: 2000
 }))
```

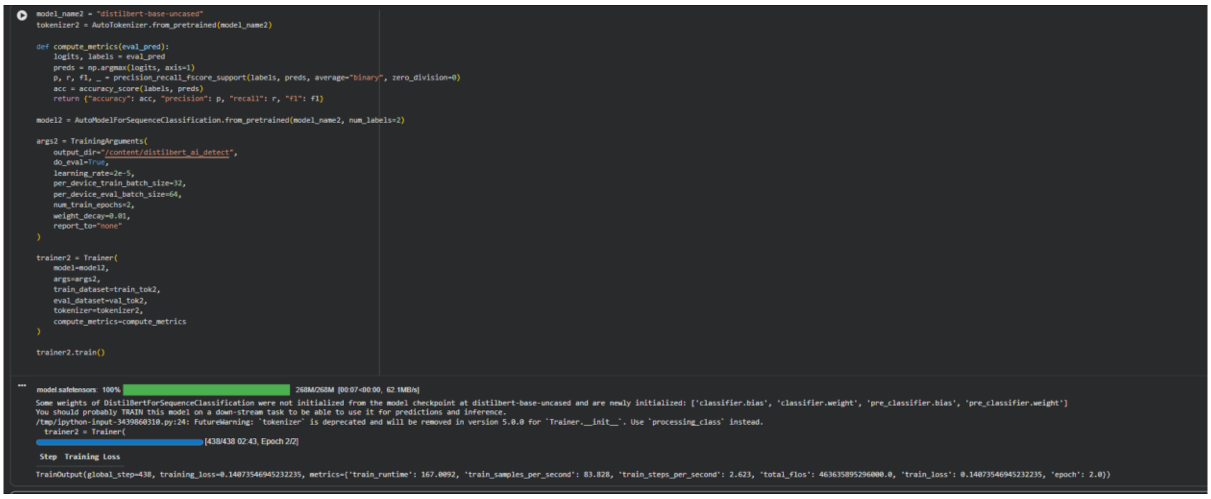


```
model_name2 = "distilbert-base-uncased"
tokenizer2 = AutoTokenizer.from_pretrained(model_name2)

max_len = 128

def tok2(batch):
    return tokenizer2(batch["text"], truncation=True, padding="max_length", max_length=max_len)

train_tok2 = train_hf.map(tok2, batched=True)
val_tok2 = val_hf.map(tok2, batched=True)
test_tok2 = test_hf.map(tok2, batched=True)

cols = ["input_ids", "attention_mask", "labels"]
train_tok2 = train_tok2.remove_columns([c for c in train_tok2.column_names if c not in cols])
val_tok2 = val_tok2.remove_columns([c for c in val_tok2.column_names if c not in cols])
test_tok2 = test_tok2.remove_columns([c for c in test_tok2.column_names if c not in cols])

train_tok2.set_format("torch")
val_tok2.set_format("torch")
test_tok2.set_format("torch")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 4.88kB/s]
config.json: 100%  483/483 [00:00<00:00, 47.1kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 525kB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 1.10MB/s]
Map: 100%  7000/7000 [00:23<00:00, 266.04 examples/s]
Map: 100%  1000/1000 [00:03<00:00, 331.45 examples/s]
Map: 100%  2000/2000 [00:08<00:00, 244.39 examples/s]
```

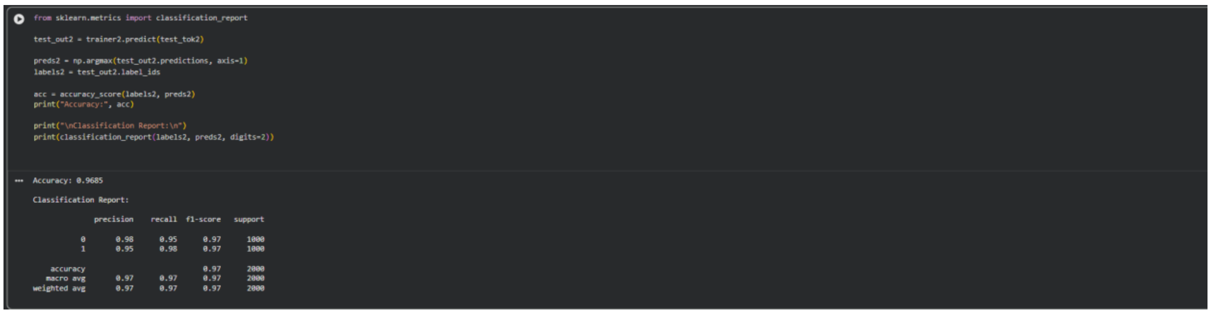


```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    p, r, f1, _ = precision_recall_fscore_support(labels, preds, average="binary", zero_division=0)
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "precision": p, "recall": r, "f1": f1}
```



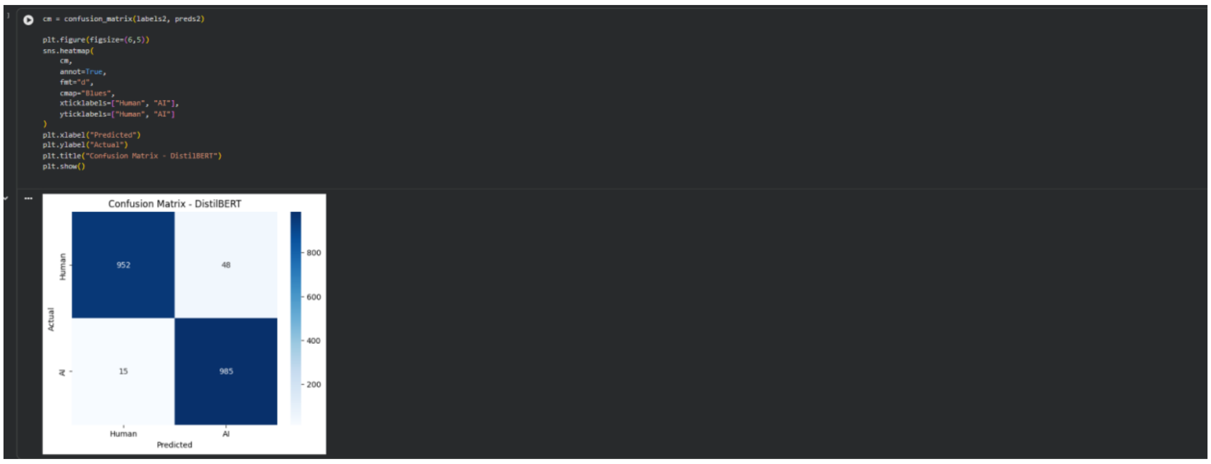
```
model_name2 = "distilbert-base-uncased"
tokenizer2 = AutoTokenizer.from_pretrained(model_name2)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    p, r, f1, _ = precision_recall_fscore_support(labels, preds, average="binary", zero_division=0)
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "precision": p, "recall": r, "f1": f1}

model2 = AutoModelForSequenceClassification.from_pretrained(model_name2, num_labels=2)

args2 = TrainingArguments(
    output_dir="/content/distilbert_ai_detect",
    do_eval=True,
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=64,
    num_train_epochs=2,
    weight_decay=0.01,
    report_to="none"
)

trainer2 = Trainer(
    model=model2,
    args=args2,
    train_dataset=train_tok2,
    eval_dataset=val_tok2,
    tokenizer=tokenizer2,
    compute_metrics=compute_metrics
)

trainer2.train()
```

```
model.safetensors: 100%  268M/268M [00:07<00:00, 62.1MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-3439860310.py:24: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer2 = Trainer(
[438/438 02:43, Epoch 2/2]
 Step  Training Loss
TrainOutput(global_step=438, training_loss=0.14073546945232235, metrics={'train_runtime': 167.0092, 'train_samples_per_second': 83.828, 'train_steps_per_second': 2.623, 'total_flos': 463635895296000.0, 'train_loss': 0.14073546945232235, 'epoch': 2.0})
```

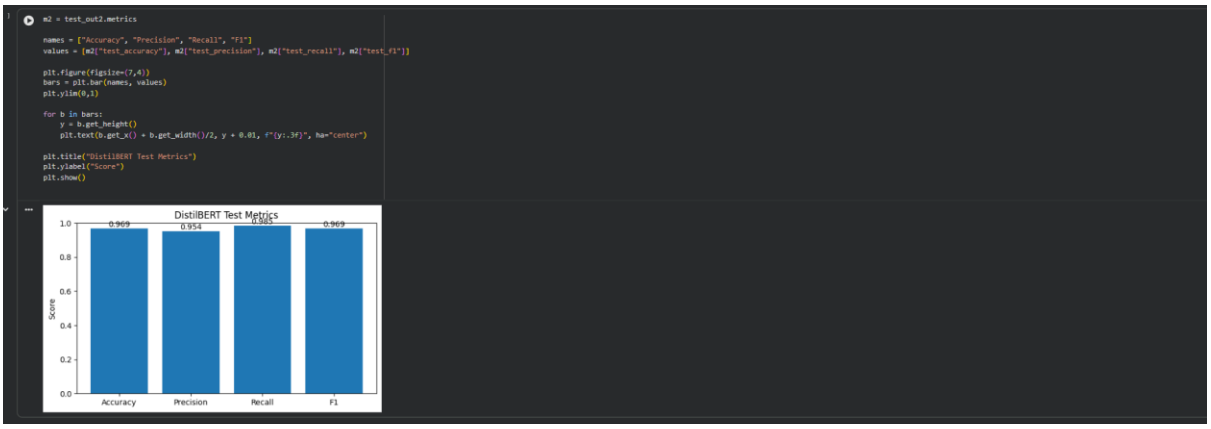


```
from sklearn.metrics import classification_report

test_out2 = trainer2.predict(test_tok2)

preds2 = np.argmax(test_out2.predictions, axis=1)
labels2 = test_out2.label_ids

acc = accuracy_score(labels2, preds2)
print("Accuracy:", acc)

print("\nClassification Report:\n")
print(classification_report(labels2, preds2, digits=2))
```

```
Accuracy: 0.9685

Classification Report:

              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1000
           1       0.95      0.98      0.97      1000

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000
```



```
cm = confusion_matrix(labels2, preds2)

plt.figure(figsize=(6,5))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Human", "AI"],
    yticklabels=["Human", "AI"]
)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - DistilBERT")
plt.show()
```



```
m2 = test_out2.metrics

names = ["Accuracy", "Precision", "Recall", "F1"]
values = [m2["test_accuracy"], m2["test_precision"], m2["test_recall"], m2["test_f1"]]

plt.figure(figsize=(7,4))
bars = plt.bar(names, values)
plt.ylim(0,1)

for b in bars:
    y = b.get_height()
    plt.text(b.get_x() + b.get_width()/2, y + 0.01, f"{y:.3f}", ha="center")

plt.title("DistilBERT Test Metrics")
plt.ylabel("Score")
plt.show()
```