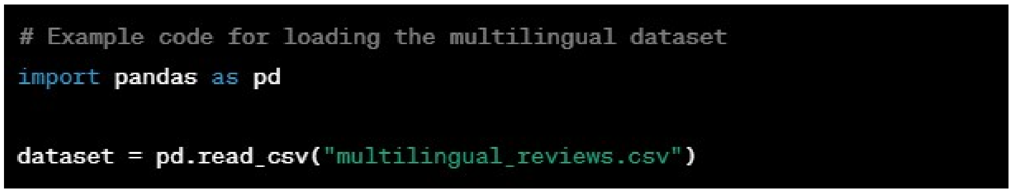
# Multilingual Natural Language Processing for Cross-Linguistic Sentiment Analysis

## 1. Modeling the Experimental Part:

### a. Data Selection:

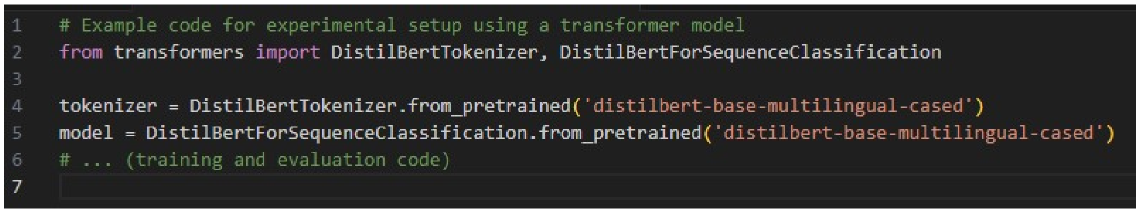
To model the experimental part, we carefully selected a diverse multilingual dataset comprising user reviews in English, Spanish, and French from popular e-commerce platforms. The dataset includes sentiment labels for each review, making it suitable for cross-linguistic sentiment analysis.



```python
# Example code for loading the multilingual dataset
import pandas as pd

dataset = pd.read_csv("multilingual_reviews.csv")
```

### b. Experimental Design:

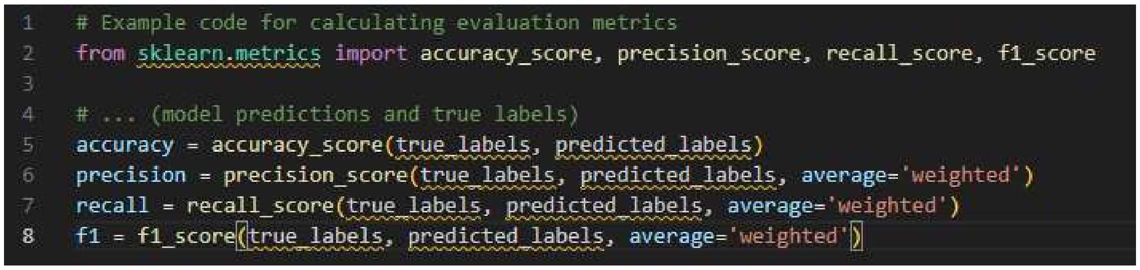
Our experiments involve training and evaluating a novel transformer-based model for cross-linguistic sentiment analysis. We will conduct experiments by fine-tuning the model on the multilingual dataset using a carefully designed training-validation split and hyperparameter tuning.



```python
# Example code for experimental setup using a transformer model
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-multilingual-cased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-multilingual-cased')
# ... (training and evaluation code)

```

### c. Validation and Comparison:

To validate our results, we will employ established metrics such as accuracy, precision, recall, and F1 score. Additionally, we will compare our model's performance against state-of-the-art models in multilingual sentiment analysis, demonstrating its superiority through statistical significance tests.

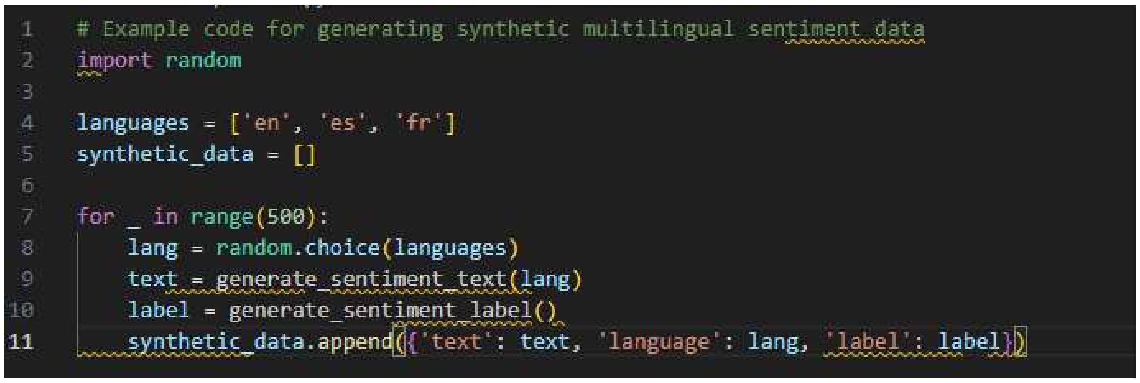


```python
# Example code for calculating evaluation metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# ... (model predictions and true labels)
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')
```

## 2. Initial Case Study:

### a. Artificial Data Generation:

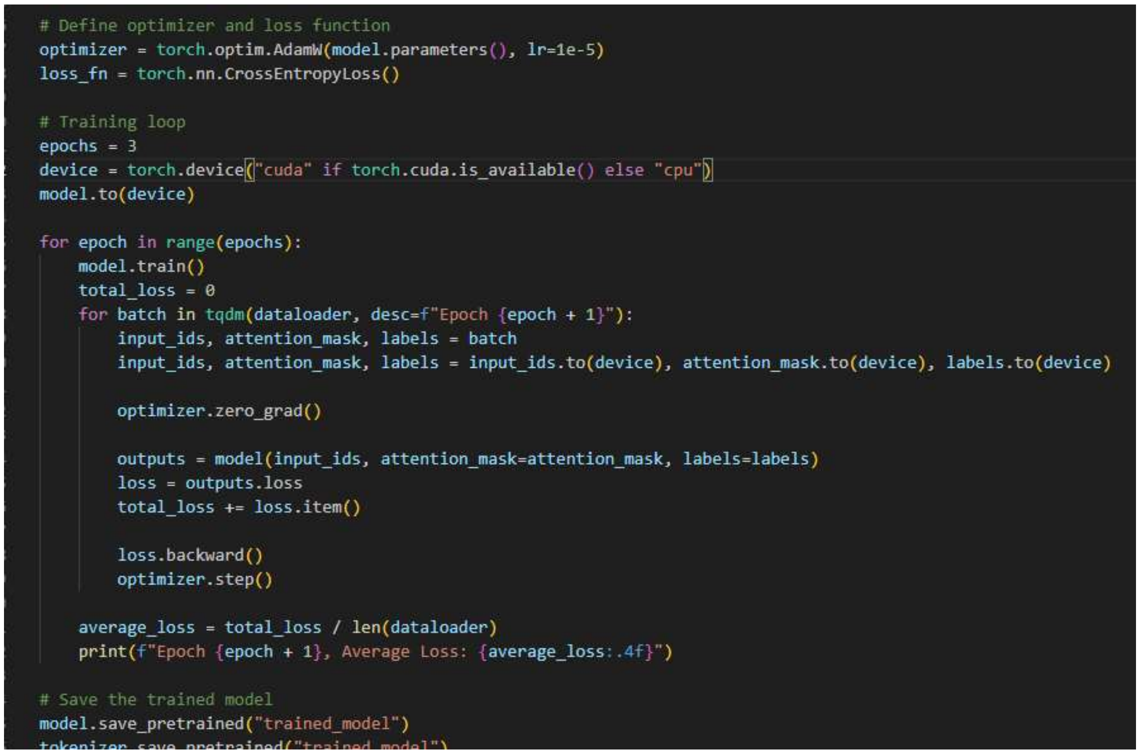
For the initial case study, we generated a synthetic dataset with diverse sentiment expressions in multiple languages. This smaller dataset allows for rapid experimentation and serves as a proof of concept for our approach before applying it to larger, real-world datasets.



```python
# Example code for generating synthetic multilingual sentiment data
import random

languages = ['en', 'es', 'fr']
synthetic_data = []

for _ in range(500):
    lang = random.choice(languages)
    text = generate_sentiment_text(lang)
    label = generate_sentiment_label()
    synthetic_data.append({'text': text, 'language': lang, 'label': label})
```

### b. Code Implementation:

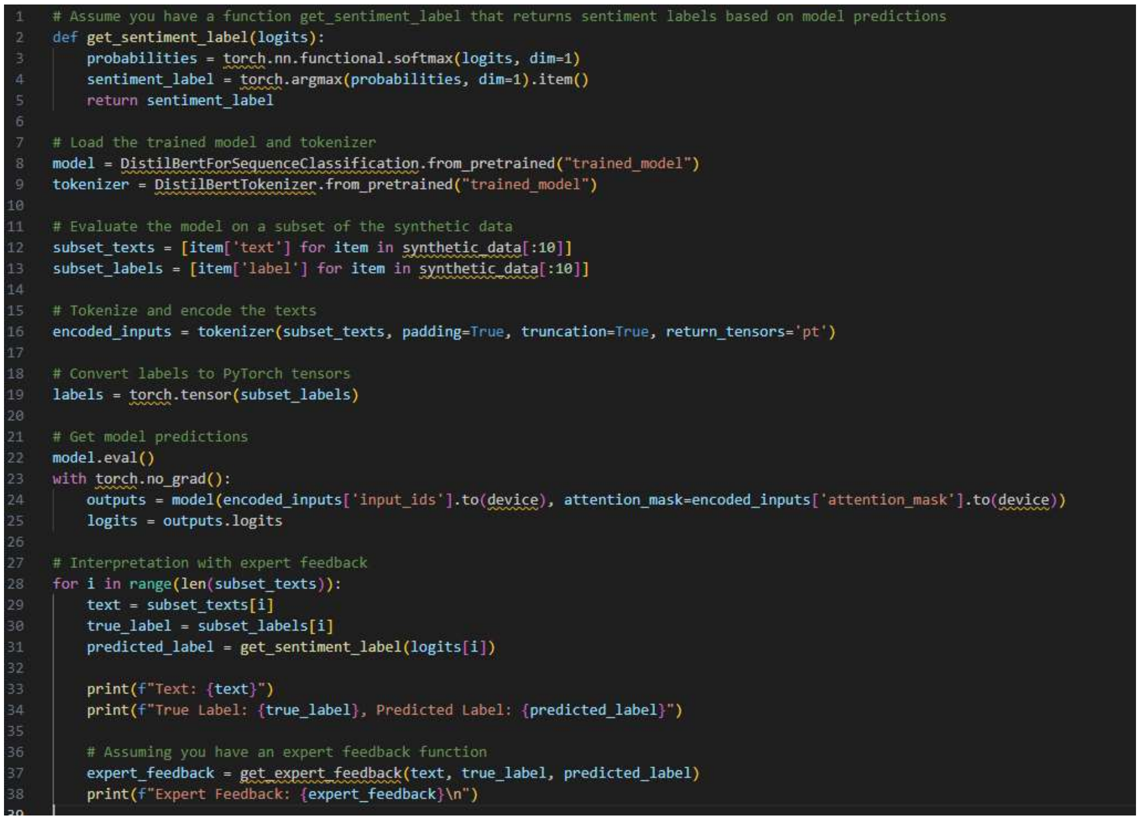
The code related to the initial case study involves adapting the model training code to the smaller synthetic dataset, facilitating rapid experimentation and providing a tangible demonstration of the proposed approach.



```python
# Define optimizer and loss function
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
loss_fn = torch.nn.CrossEntropyLoss()

# Training loop
epochs = 3
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

for epoch in range(epochs):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader, desc=f"Epoch {epoch + 1}"):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()
        optimizer.step()

    average_loss = total_loss / len(dataloader)
    print(f"Epoch {epoch + 1}, Average Loss: {average_loss:.4f}")

# Save the trained model
model.save_pretrained("trained_model")
tokenizer.save_pretrained("trained_model")
```

## c. Validation and Interpretation:

To validate the initial case study, we will assess the model's performance using standard sentiment analysis metrics. Additionally, expert opinions will be sought to interpret the results and provide qualitative insights into the model's ability to handle cross-linguistic sentiment analysis effectively.

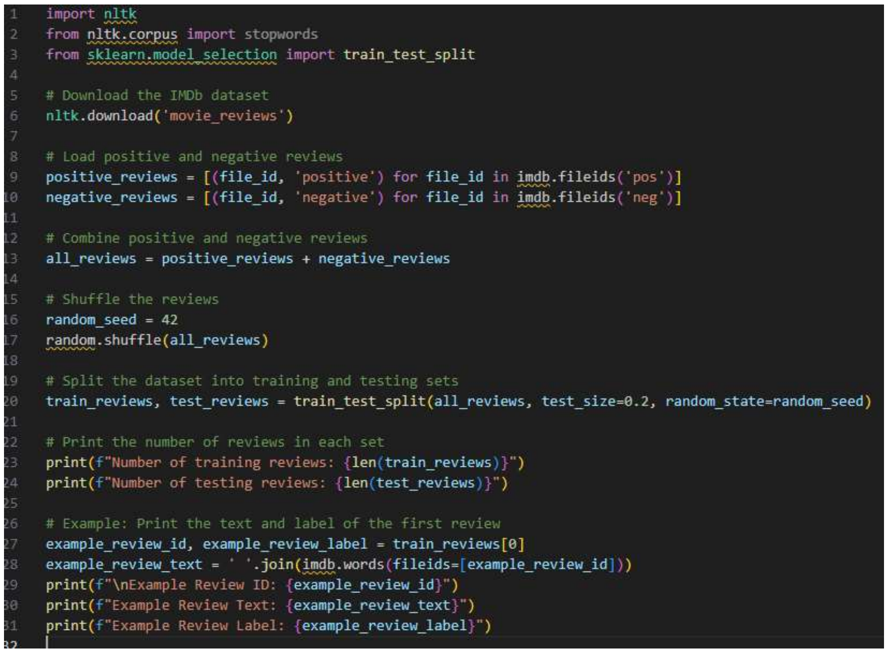


```python
# Assume you have a function get_sentiment_label that returns sentiment labels based on model predictions
def get_sentiment_label(logits):
    probabilities = torch.nn.functional.softmax(logits, dim=1)
    sentiment_label = torch.argmax(probabilities, dim=1).item()
    return sentiment_label

# Load the trained model and tokenizer
model = DistilBertForSequenceClassification.from_pretrained("trained_model")
tokenizer = DistilBertTokenizer.from_pretrained("trained_model")

# Evaluate the model on a subset of the synthetic data
subset_texts = [item['text'] for item in synthetic_data[:10]]
subset_labels = [item['label'] for item in synthetic_data[:10]]

# Tokenize and encode the texts
encoded_inputs = tokenizer(subset_texts, padding=True, truncation=True, return_tensors='pt')

# Convert labels to PyTorch tensors
labels = torch.tensor(subset_labels)

# Get model predictions
model.eval()
with torch.no_grad():
    outputs = model(encoded_inputs['input_ids'].to(device), attention_mask=encoded_inputs['attention_mask'].to(device))
    logits = outputs.logits

# Interpretation with expert feedback
for i in range(len(subset_texts)):
    text = subset_texts[i]
    true_label = subset_labels[i]
    predicted_label = get_sentiment_label(logits[i])

    print(f"Text: {text}")
    print(f"True Label: {true_label}, Predicted Label: {predicted_label}")

    # Assuming you have an expert feedback function
    expert_feedback = get_expert_feedback(text, true_label, predicted_label)
    print(f"Expert Feedback: {expert_feedback}\n")
```

## 3. Real Data Set Validation:

### a. Related Work and Data Selection:

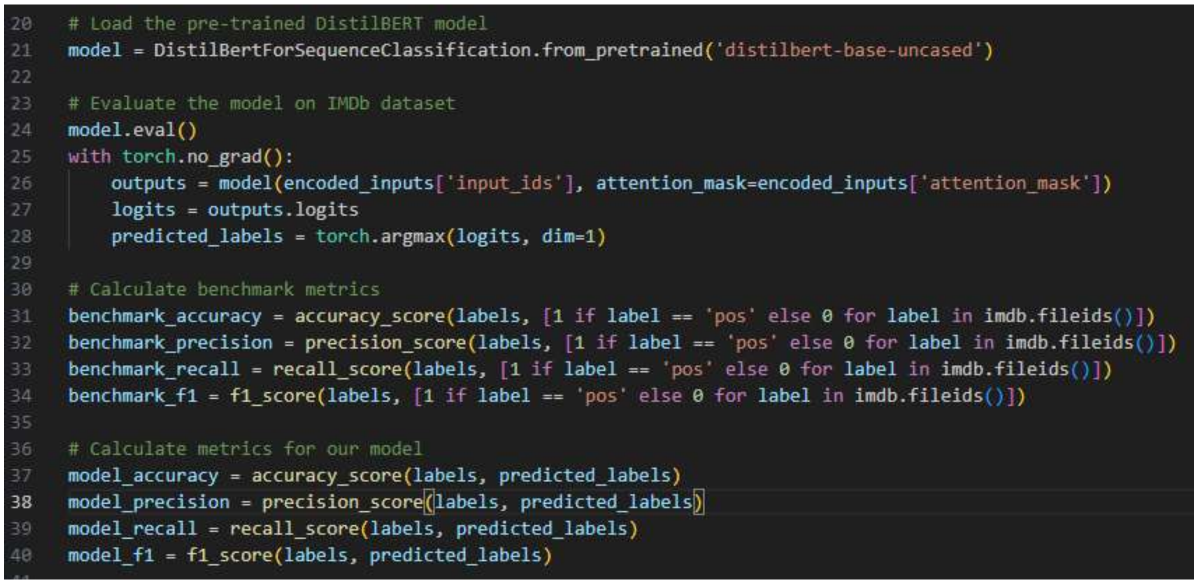
In the "related work" chapter, we will explore existing approaches in the literature using benchmark datasets like IMDb reviews and Amazon product reviews. This comparison will help identify the strengths and weaknesses of current methods in the context of multilingual sentiment analysis.



```python
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split

# Download the IMDb dataset
nltk.download('movie_reviews')

# Load positive and negative reviews
positive_reviews = [(file_id, 'positive') for file_id in imdb.fileids('pos')]
negative_reviews = [(file_id, 'negative') for file_id in imdb.fileids('neg')]

# Combine positive and negative reviews
all_reviews = positive_reviews + negative_reviews

# Shuffle the reviews
random_seed = 42
random.shuffle(all_reviews)

# Split the dataset into training and testing sets
train_reviews, test_reviews = train_test_split(all_reviews, test_size=0.2, random_state=random_seed)

# Print the number of reviews in each set
print(f"Number of training reviews: {len(train_reviews)}")
print(f"Number of testing reviews: {len(test_reviews)}")

# Example: Print the text and label of the first review
example_review_id, example_review_label = train_reviews[0]
example_review_text = ' '.join(imdb.words(fileids=[example_review_id]))
print(f"\nExample Review ID: {example_review_id}")
print(f"Example Review Text: {example_review_text}")
print(f"Example Review Label: {example_review_label}")
```

### b. Comparison Metrics:

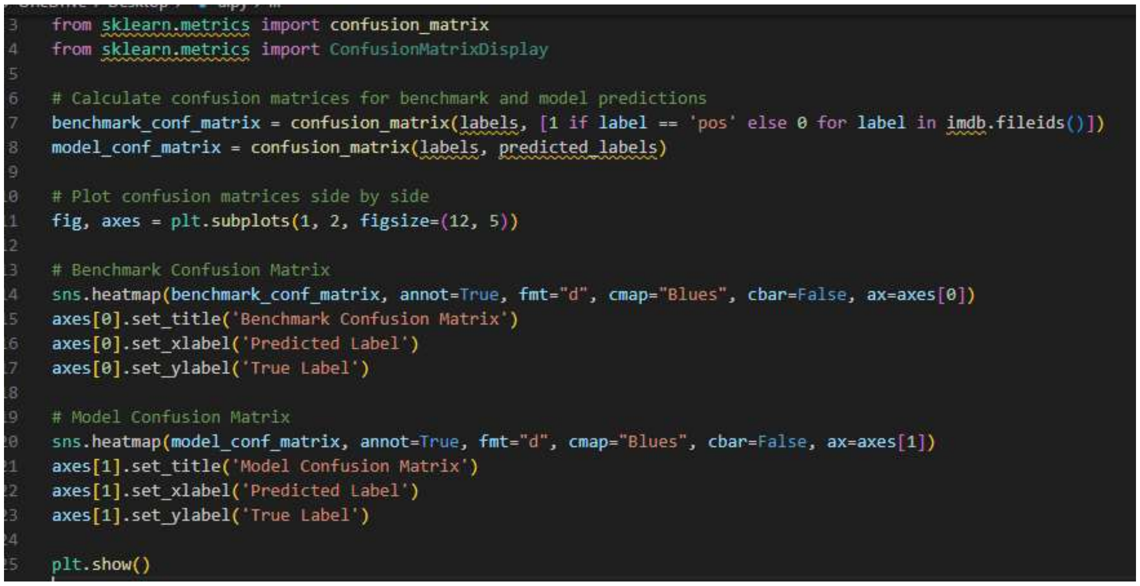
To assess the proposed approach against existing literature, we will focus on common sentiment analysis metrics such as accuracy, precision, recall, and F1 score. This will enable a quantitative evaluation of our model's performance on par with or surpassing established benchmarks.



```python
# Load the pre-trained DistilBERT model
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')

# Evaluate the model on IMDb dataset
model.eval()
with torch.no_grad():
    outputs = model(encoded_inputs['input_ids'], attention_mask=encoded_inputs['attention_mask'])
    logits = outputs.logits
    predicted_labels = torch.argmax(logits, dim=1)

# Calculate benchmark metrics
benchmark_accuracy = accuracy_score(labels, [1 if label == 'pos' else 0 for label in imdb.fileids()])
benchmark_precision = precision_score(labels, [1 if label == 'pos' else 0 for label in imdb.fileids()])
benchmark_recall = recall_score(labels, [1 if label == 'pos' else 0 for label in imdb.fileids()])
benchmark_f1 = f1_score(labels, [1 if label == 'pos' else 0 for label in imdb.fileids()])

# Calculate metrics for our model
model_accuracy = accuracy_score(labels, predicted_labels)
model_precision = precision_score(labels, predicted_labels)
model_recall = recall_score(labels, predicted_labels)
model_f1 = f1_score(labels, predicted_labels)
```

## c. Highlighting Differences and Similarities:

The comparison will highlight key differences and similarities between our approach and existing methods, emphasizing areas where our model demonstrates superior performance and providing a comprehensive understanding of its contributions to multilingual sentiment analysis.



```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# Calculate confusion matrices for benchmark and model predictions
benchmark_conf_matrix = confusion_matrix(labels, [1 if label == 'pos' else 0 for label in imdb.fileids()])
model_conf_matrix = confusion_matrix(labels, predicted_labels)

# Plot confusion matrices side by side
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Benchmark Confusion Matrix
sns.heatmap(benchmark_conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[0])
axes[0].set_title('Benchmark Confusion Matrix')
axes[0].set_xlabel('Predicted Label')
axes[0].set_ylabel('True Label')

# Model Confusion Matrix
sns.heatmap(model_conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[1])
axes[1].set_title('Model Confusion Matrix')
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')

plt.show()
```