Configuration Manual

MSc Research Project
Data Analytics

Sathish Omega Suresh
Student ID: x21228388

School of Computing
National College of Ireland

Supervisor: Teerath Kumar Menghwar

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sathish Omega Suresh
Student ID:	21228388
Programme:	MSc. in Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Teerath Kumar Menghwar
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	810
Page Count:	25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sathish Omega Suresh
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sathish Omega Suresh
X21228388

1 Introduction

The configuration manual outlines the orderly, step-by-step instructions for executing the research project's related sections and the procedures for evaluating them. The instructions include a number of requirements, ranging from the installation of applications to the creation of a model. Identifying the emotions for the questionText using the conversational dataset and using the defined BERT model, which is utilized for text classification coupled with the similarity algorithm, are two different stages of this project. In the parts that follow, specific code snippets for carrying out the same task are provided.

2 System Configuration

2.1 System Configuration

The study project was created utilizing Google Colab, an open-source platform for AI/ML projects in the Google ecosystem, as well as the free IDE Jupyter Notebook. This setting is powered by a Python module. Installing each of these packages is necessary before the project can be built.

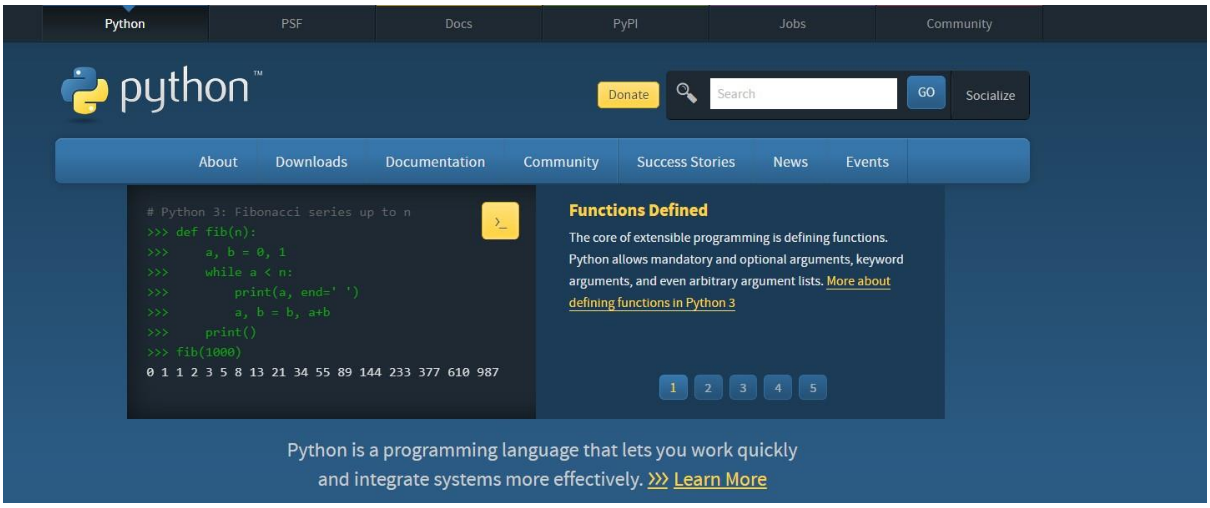
2.2 Hardware specifications

- System Name: LAPTOP-CM08LV4S
- Processor: AMD Ryzen 7 4800H with Radeon Graphics - 2.90 GHz
- Installed RAM: 16.00 GB
- Storage Size: 1TB SSD (109,951,162,7776 bytes)
- OS type: 64-bit operating system, x64-based processor

3 Installation and Environment Setup

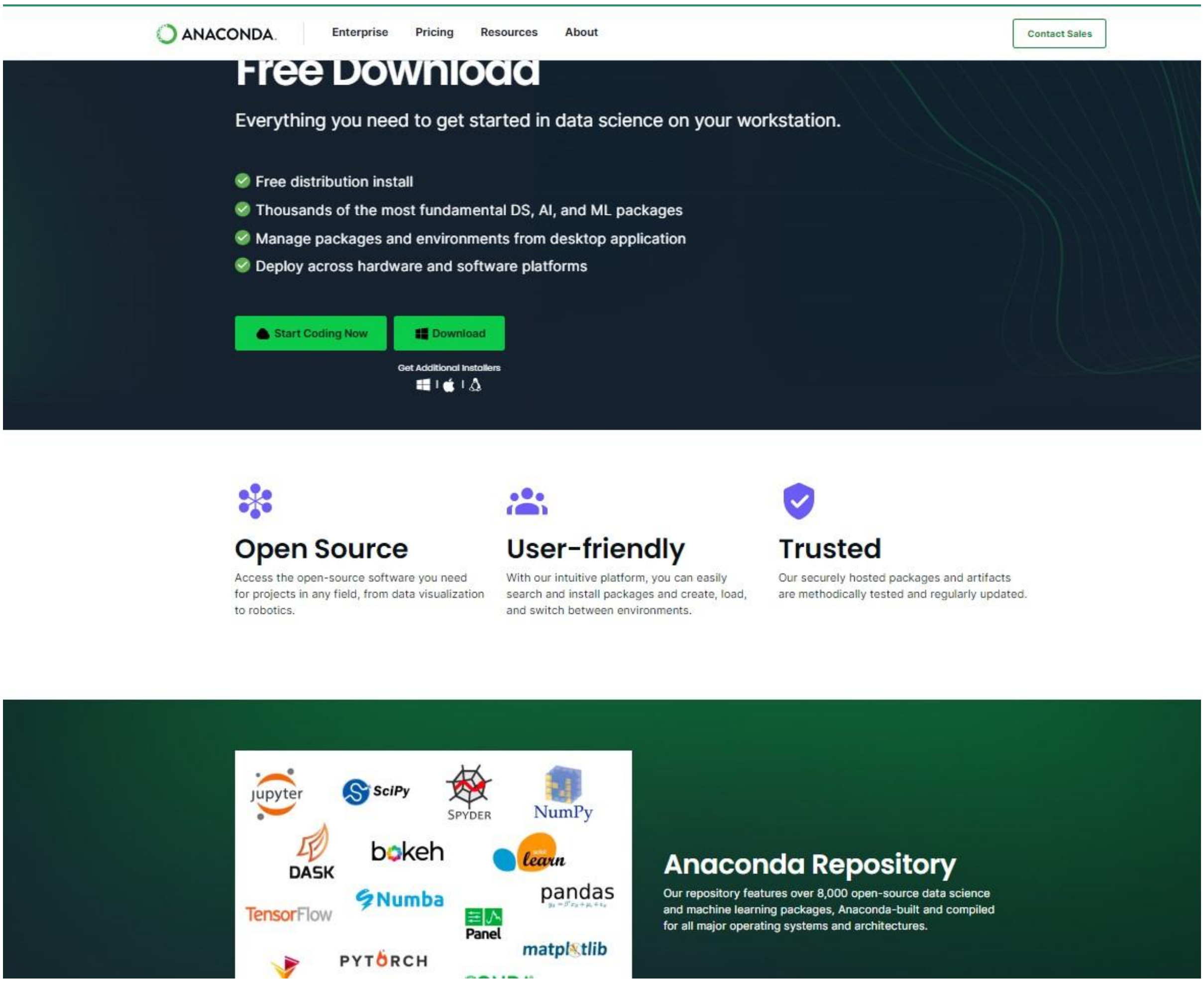
- **Python**

This project made use of a Python package. Since the majority of Deep Learning and Machine Learning Projects are supported by its numerous built-in libraries. With a variety of plots, it makes developing and analysing models easier. Installing the most recent version of Python on the machine is the first prerequisite. The package installer is capable of being downloaded through a web browser from the website reference https://www.python.org/downloads depending on the operating system. Type 'python -version' in the command prompt to confirm Python has been successfully installed from the website, as shown in figure python below.



- **Anaconda**

The anaconda package includes a number of IDE that are helpful for writing code and analyzing outputs from python packages. As seen in the below figure, this package can be obtained and installed from the website https://www.anaconda.com/products/individual. Jupyter notebook and its tasks are launched in browser tabs from the anaconda navigator. Python notebooks are first created and saved in the.ipynb format.



- **Jupyter Notebook**

Using the pip command, the python libraries are installed during the execution of code. Transformers, Scikit-Learn, nltk, Numpy, Pandas, Tensorflow, Matplotlib, googletrans, Seaborn, and Plotly are the necessary libraries for this course of action. In this browser, many different IDEs were available. The model in this project is constructed in Jupyter Notebook.

Command: pip install 'LibraryName'

4 Data Collection

There is one dataset used for this project which was semantically developed with chat instances based on different scenarios. Following sections where the data sets of a conversational excel file are being contained into a variable for preprocessing as shown in the below figure. These are used in the respective image and text processing models, which is concatenated at the end yield an output used to satisfy the research objectives.

5 Implementation

5.1 Importing Libraries

The implementation part is explained below in detail on how the project was implemented using Python. Please carry out the instructions step by step. The first step is to preprocess the provided data before we start the implementation. The libraries required for startup are displayed in the below picture.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import LabelEncoder
from sentence_transformers import SentenceTransformer
import torch
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from transformers import pipeline
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Import & load the data in a data frame

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

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

```
# Load the dataset from Excel
data = pd.read_excel("/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/emp_burnout.xlsx")

# Visualize the first few rows of the dataset
print(data.head())
```

```
   qnID                                       questionText  \
0     1  Hey, I've been feeling really overwhelmed and ...
1     2  Well, the workload has been increasing, and th...
2     3  Not yet. I'm afraid they'll see it as a weakne...
3     4  That makes sense. I guess I just need to find ...
4     5  Thank you for the advice. I'll try to have tha...

                                          answerText          emotions
0  I'm glad you reached out. It's important to ad...       Overwhelmed
1  I understand how challenging that can be. It s...           Anxious
2  It's important to remember that asking for sup...           Fearful
3  Absolutely. Start by scheduling a meeting with...  Seeking guidance
4  Certainly. It's important to prioritize self-c...    Seeking advice
```

5.2 Data Preprocessing and Data augmentation

5.2.1 Data Preprocessing

The preprocessing on the given data containing the excel file is performed as shown in the figure 5 below,

```
# Clean and preprocess text data
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespaces
    return text

data['questionText'] = data['questionText'].apply(preprocess_text)

# Lemmatize
def lemmatize(text):
    lemmatizer = WordNetLemmatizer()
    words = word_tokenize(text)  # Tokenize the text
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(lemmatized_words)

data['questionText'] = data['questionText'].apply(lemmatize)

# Remove stopwords
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    filtered_words = [word for word in text.split() if word not in stop_words]
    return ' '.join(filtered_words)

data['questionText'] = data['questionText'].apply(remove_stopwords)
```

The pre-processed data is also considered for data augmentation for model fitting. Finally, saving the augmented data in a file path with an additional labels encoder for mapping the emotions column from the data frame for further use in the study.

```
# Synonym Replacement
def synonym_replacement(text):
    words = word_tokenize(text)
    new_words = []
    for word in words:
        synonyms = wordnet.synsets(word)
        if synonyms:
            synonym = synonyms[0].lemmas()[0].name()
            new_words.append(synonym)
        else:
            new_words.append(word)
    return ' '.join(new_words)

def rephrase_question(text):
    tokens = word_tokenize(text)
    rephrased_tokens = [synonym_replacement(token) for token in tokens]
    return ' '.join(rephrased_tokens)

# Dialogue combination
def dialogue_combination(text, num_samples=1):
    augmented_data = []
    for _ in range(num_samples):
        indexes = random.sample(range(len(data)), 2)
        question1, answer1 = data.loc[indexes[0]]
        question2, answer2 = data.loc[indexes[1]]
        new_question = f"{question1} {question2}"
        new_answer = f"{answer1} {answer2}"
        augmented_data.append((new_question, new_answer))
    return augmented_data
```

```
# Paraphrasing using Google Translate (English to Spanish and back to English)
def paraphrasing(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='es')
    paraphrased = translator.translate(translation.text, src='es', dest='en')
    return paraphrased.text

# Back-translation using Google Translate (English to French and back to English)
def back_translation(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='fr')
    back_translated = translator.translate(translation.text, src='fr', dest='en')
    return back_translated.text

# Augment the dataframe using data augmentation techniques
augmented_data = []
```

```
for index, row in data.iterrows():
    question = row['questionText']
    answer = row['answerText']
    emotion = row['emotions']

    # Original data
    augmented_data.append({'questionText': question, 'emotions': emotion, 'answerText': answer})

    # Synonym Replacement
    augmented_data.append({'questionText': synonym_replacement(question), 'emotions': emotion, 'answerText': answer})

    # rephrase_question
    augmented_data.append({'questionText': rephrase_question(question), 'emotions': emotion, 'answerText': answer})

    # Paraphrasing
    augmented_data.append({'questionText': paraphrasing(question), 'emotions': emotion, 'answerText': answer})

    # Back-translation
    augmented_data.append({'questionText': back_translation(question), 'emotions': emotion, 'answerText': answer})

# Create augmented dataframe
augmented_df = pd.DataFrame(augmented_data)

# Display augmented dataframe
print(augmented_df)
```

```
# Encoding emotions into numerical labels using LabelEncoder
label_encoder = LabelEncoder()
preprocessed_data['emotions_encoded'] = label_encoder.fit_transform(preprocessed_data['emotions'])
print (preprocessed_data)
```

```
                                           questionText      emotions  \
0     hey ive feeling really overwhelmed stressed la...   Overwhelmed
1     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
2     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
3     Hey, I feel very overwhelmed stressed lately, ...   Overwhelmed
4     Hey, I feel really overwhelmed in recent times...   Overwhelmed
...                                                 ...           ...
1475  thank suggestion ill make effort implement fin...  Appreciative
1476  thank suggestion ailment brand attempt impleme...  Appreciative
1477  thank suggestion ailment brand attempt impleme...  Appreciative
1478  Appreciate suggestion.I will make the effort i...  Appreciative
1479  thank you suggestion badly making efforts to i...  Appreciative

                                             answerText  emotions_encoded
0     I'm glad you reached out. It's important to ad...                38
1     I'm glad you reached out. It's important to ad...                38
2     I'm glad you reached out. It's important to ad...                38
3     I'm glad you reached out. It's important to ad...                38
4     I'm glad you reached out. It's important to ad...                38
...                                                 ...               ...
1475  You're welcome. Remember, seeking support is a...                 5
1476  You're welcome. Remember, seeking support is a...                 5
1477  You're welcome. Remember, seeking support is a...                 5
1478  You're welcome. Remember, seeking support is a...                 5
1479  You're welcome. Remember, seeking support is a...                 5

[1480 rows x 4 columns]
```

6 Model Building

6.1 Implementing BERT Model

The below code gives the overview of model building, setting hyper tuning parameters of the BERT model for text classification. Load the predefined BERT model, and define the constants and parameters.

```
# Define constants
NUM_EMOTIONS = 78
MODEL_NAME = "bert-base-uncased"
BATCH_SIZE = 16
MAX_LEN = 512
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# Load the pre-trained BERT model and tokenizer
tokenizer = BertTokenizerFast.from_pretrained(MODEL_NAME)
model = BertForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=NUM_EMOTIONS).to(DEVICE)

class EmotionDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return {
            'input_ids': torch.tensor(self.encodings.input_ids[idx]),
            'attention_mask': torch.tensor(self.encodings.attention_mask[idx]),
            'labels': torch.tensor(self.labels[idx])
        }
```

6.2 Splitting of Train and Test Data

The given data set comprises of conversational text data , which is considered for modelling now , let us split the dataset into training and testing sets and convert them to BERT format as shown in the Figure below,

```
# Split the dataset into training and testing sets
train_text, test_text, train_labels, test_labels = train_test_split(preprocessed_data['questionText'].tolist(),
                                                                    preprocessed_data['emotions_encoded'].tolist(),
                                                                    test_size=0.2, random_state=42)

# Convert the text to BERT format
train_encodings = tokenizer(train_text, truncation=True, padding=True, max_length=MAX_LEN)
test_encodings = tokenizer(test_text, truncation=True, padding=True, max_length=MAX_LEN)
```

then create dataset objects and also create data loaders for both train and test datasets.

```python
# Create Dataset objects
train_dataset = EmotionDataset(train_encodings, train_labels)
test_dataset = EmotionDataset(test_encodings, test_labels)

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

also, import the the necessary libraries for BERT optimized modelling.

```python
import torch.optim as optim
import torch.nn as nn
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score
from transformers import get_linear_schedule_with_warmup
```

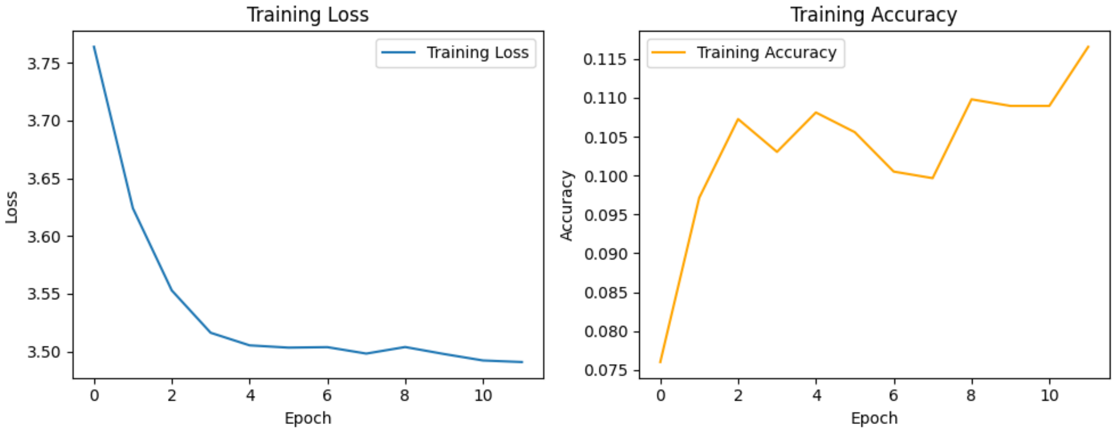
```python
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

```python
# Lists to store losses and accuracies
train_losses = []
train_accuracies = []
```

```python
NUM_EPOCHS = 10

# Define the optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=len(train_loader) * NUM_EPOCHS
)

# Training loop
for epoch in range(NUM_EPOCHS):
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_predictions = 0
    for batch in train_loader:
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)
        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        scheduler.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.logits.data, 1)
        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = correct_predictions / total_predictions

    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy)

    print(f"Epoch [{epoch+1}/{NUM_EPOCHS}] - Loss: {epoch_loss:.4f} - Accuracy: {epoch_accuracy:.4f}")
```

```python
# Save the model
model_path = "/content/drive/MyDrive/MasterThesisChatBot/EMP_CB"
model.save_pretrained(model_path)
tokenizer.save_pretrained(model_path)

# Plot the accuracy and loss graphs
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy', color='orange')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is deprecated and
  warnings.warn(
Epoch [1/10] - Loss: 3.5529 - Accuracy: 0.1073
Epoch [2/10] - Loss: 3.5162 - Accuracy: 0.1030
Epoch [3/10] - Loss: 3.5053 - Accuracy: 0.1081
Epoch [4/10] - Loss: 3.5033 - Accuracy: 0.1056
Epoch [5/10] - Loss: 3.5037 - Accuracy: 0.1005
Epoch [6/10] - Loss: 3.4982 - Accuracy: 0.0997
Epoch [7/10] - Loss: 3.5038 - Accuracy: 0.1098
Epoch [8/10] - Loss: 3.4978 - Accuracy: 0.1090
Epoch [9/10] - Loss: 3.4922 - Accuracy: 0.1090
Epoch [10/10] - Loss: 3.4908 - Accuracy: 0.1166
```



```
# Set the model to evaluation mode
model.eval()

correct_predictions = 0
total_predictions = 0

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits.data, 1)

        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

accuracy = correct_predictions / total_predictions
accuracy_percentage = accuracy * 100

print(f"Test Accuracy: {accuracy_percentage:.2f}%")
```

```
Test Accuracy: 11.82%
```

6.3 Fine Tuning the BERT Algorithm

```
#Additional Preprocess

from transformers import BertTokenizer

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Replace this with your actual DataFrame loading code
prepdata = pd.read_excel('/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/augmented_data.xlsx')

# Preprocess and tokenize the 'questionText' column
def tokenize_text(text):
    tokens = tokenizer.encode_plus(
        text,
        add_special_tokens=True,  # Add [CLS] and [SEP] tokens
        max_length=128,           # Adjust this based on your sequence length
        pad_to_max_length=True,
        return_attention_mask=True,
        return_tensors='pt'       # Return PyTorch tensors
    )
    return tokens

# Apply the tokenizer function to the 'questionText' column
prepdata['tokenized_question'] = prepdata['questionText'].apply(tokenize_text)

print(prepdata)
```

```
                                           questionText      emotions  \
0     hey ive feeling really overwhelmed stressed la...   Overwhelmed
1     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
2     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
3     Hey, I feel very overwhelmed stressed lately, ...   Overwhelmed
4     Hey, I feel really overwhelmed in recent times...   Overwhelmed
...                                                 ...           ...
1475  thank suggestion ill make effort implement fin...  Appreciative
1476  thank suggestion ailment brand attempt impleme...  Appreciative
1477  thank suggestion ailment brand attempt impleme...  Appreciative
1478  Appreciate suggestion.I will make the effort i...  Appreciative
1479  thank you suggestion badly making efforts to i...  Appreciative

                                             answerText  \
0     I'm glad you reached out. It's important to ad...
1     I'm glad you reached out. It's important to ad...
2     I'm glad you reached out. It's important to ad...
3     I'm glad you reached out. It's important to ad...
4     I'm glad you reached out. It's important to ad...
...                                                 ...
1475  You're welcome. Remember, seeking support is a...
1476  You're welcome. Remember, seeking support is a...
1477  You're welcome. Remember, seeking support is a...
1478  You're welcome. Remember, seeking support is a...
1479  You're welcome. Remember, seeking support is a...

                               tokenized_question
0     [input_ids, token_type_ids, attention_mask]
1     [input_ids, token_type_ids, attention_mask]
2     [input_ids, token_type_ids, attention_mask]
3     [input_ids, token_type_ids, attention_mask]
4     [input_ids, token_type_ids, attention_mask]
...                                           ...
1475  [input_ids, token_type_ids, attention_mask]
1476  [input_ids, token_type_ids, attention_mask]
1477  [input_ids, token_type_ids, attention_mask]
1478  [input_ids, token_type_ids, attention_mask]
1479  [input_ids, token_type_ids, attention_mask]

[1480 rows x 4 columns]
```

```
# Encoding emotions into numerical labels using LabelEncoder
label_encoder = LabelEncoder()
prepdata['emotions_encoded'] = label_encoder.fit_transform(prepdata['emotions'])
print (prepdata)
```

```
                                           questionText      emotions  \
0     hey ive feeling really overwhelmed stressed la...   Overwhelmed
1     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
2     hey ive feeling truly overwhelm stress recentl...   Overwhelmed
3     Hey, I feel very overwhelmed stressed lately, ...   Overwhelmed
4     Hey, I feel really overwhelmed in recent times...   Overwhelmed
...                                                 ...           ...
1475  thank suggestion ill make effort implement fin...  Appreciative
1476  thank suggestion ailment brand attempt impleme...  Appreciative
1477  thank suggestion ailment brand attempt impleme...  Appreciative
1478  Appreciate suggestion.I will make the effort i...  Appreciative
1479  thank you suggestion badly making efforts to i...  Appreciative

                                             answerText  \
0     I'm glad you reached out. It's important to ad...
1     I'm glad you reached out. It's important to ad...
2     I'm glad you reached out. It's important to ad...
3     I'm glad you reached out. It's important to ad...
4     I'm glad you reached out. It's important to ad...
...                                                 ...
1475  You're welcome. Remember, seeking support is a...
1476  You're welcome. Remember, seeking support is a...
1477  You're welcome. Remember, seeking support is a...
1478  You're welcome. Remember, seeking support is a...
1479  You're welcome. Remember, seeking support is a...

                               tokenized_question  emotions_encoded
0     [input_ids, token_type_ids, attention_mask]                38
1     [input_ids, token_type_ids, attention_mask]                38
2     [input_ids, token_type_ids, attention_mask]                38
3     [input_ids, token_type_ids, attention_mask]                38
4     [input_ids, token_type_ids, attention_mask]                38
...                                           ...               ...
1475  [input_ids, token_type_ids, attention_mask]                 5
1476  [input_ids, token_type_ids, attention_mask]                 5
1477  [input_ids, token_type_ids, attention_mask]                 5
1478  [input_ids, token_type_ids, attention_mask]                 5
1479  [input_ids, token_type_ids, attention_mask]                 5

[1480 rows x 5 columns]
```

```python
# Define constants
NUM_EMOTIONS = 78
MODEL_NAME = "bert-base-uncased"
BATCH_SIZE = 16
MAX_LEN = 512
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the pre-trained BERT model and tokenizer
tokenizer = BertTokenizerFast.from_pretrained(MODEL_NAME)
finetune_model = BertForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=NUM_EMOTIONS).to(DEVICE)
```

```python
# Define your dataset class
class CustomDataset(Dataset):
    def __init__(self, prepdata, tokenizer, max_length):
        self.prepdata = prepdata
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.prepdata)

    def __getitem__(self, idx):
        item = self.prepdata.iloc[idx]
        question = str(item['questionText'])
        emotions = int(item['emotions_encoded'])  # Assuming you have an 'emotions' column in your DataFrame

        inputs = self.tokenizer.encode_plus(
            question,
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            return_tensors='pt',
            truncation=True
        )

        return {
            'input_ids': inputs['input_ids'].flatten(),
            'attention_mask': inputs['attention_mask'].flatten(),
            'labels': torch.tensor(emotions, dtype=torch.long)
        }

# Create the dataset and data loader
MAX_LENGTH = 128  # Set your desired maximum sequence length
train_dataset = CustomDataset(prepdata, tokenizer, max_length=MAX_LENGTH)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

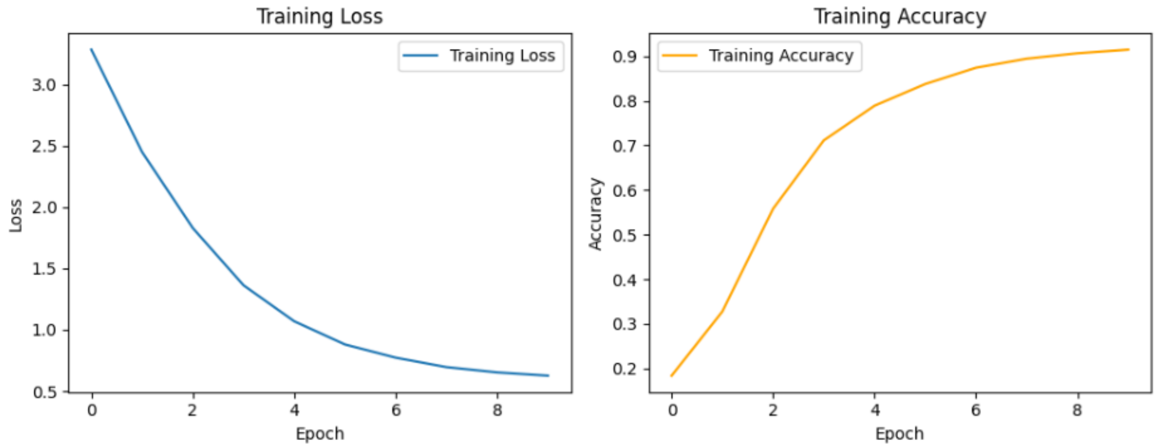
```python
# Define other necessary variables
NUM_EPOCHS = 10

# Define the optimizer and learning rate scheduler
optimizer = AdamW(finetune_model.parameters(), lr=2e-5, correct_bias=False)
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=len(train_loader) * NUM_EPOCHS
)

train_losses = []
train_accuracies = []
```

```
# Training loop
for epoch in range(NUM_EPOCHS):
    finetune_model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_predictions = 0

    for batch in train_loader:
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        optimizer.zero_grad()
        outputs = finetune_model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        scheduler.step()

        running_loss += loss.item()

        _, predicted = torch.max(outputs.logits.data, 1)
        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = correct_predictions / total_predictions

    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy)

    print(f"Epoch [{epoch+1}/{NUM_EPOCHS}] - Loss: {epoch_loss:.4f} - Accuracy: {epoch_accuracy:.4f}")


# Save the fine-tuned model
finemodel_path = "/content/drive/MyDrive/MasterThesisChatBot/EMP_CB_finetune"
finetune_model.save_pretrained(finemodel_path)
tokenizer.save_pretrained(finemodel_path)
```


```
# Save the fine-tuned model
finemodel_path = "/content/drive/MyDrive/MasterThesisChatBot/EMP_CB_finetune"
finetune_model.save_pretrained(finemodel_path)
tokenizer.save_pretrained(finemodel_path)
```


```
# Plot the accuracy and loss graphs
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy', color='orange')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is deprecated
  warnings.warn(
Epoch [1/10] - Loss: 3.2823 - Accuracy: 0.1838
Epoch [2/10] - Loss: 2.4485 - Accuracy: 0.3277
Epoch [3/10] - Loss: 1.8289 - Accuracy: 0.5581
Epoch [4/10] - Loss: 1.3624 - Accuracy: 0.7115
Epoch [5/10] - Loss: 1.0691 - Accuracy: 0.7892
Epoch [6/10] - Loss: 0.8801 - Accuracy: 0.8378
Epoch [7/10] - Loss: 0.7727 - Accuracy: 0.8743
Epoch [8/10] - Loss: 0.6956 - Accuracy: 0.8946
Epoch [9/10] - Loss: 0.6525 - Accuracy: 0.9068
Epoch [10/10] - Loss: 0.6260 - Accuracy: 0.9149
```




```
 # Set the model to evaluation mode
finetune_model.eval()

correct_predictions = 0
total_predictions = 0

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        outputs = finetune_model(input_ids=input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits.data, 1)

        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

accuracy = correct_predictions / total_predictions
accuracy_percentage = accuracy * 100

print(f"Test Accuracy: {accuracy_percentage:.2f}%")
```

```
Test Accuracy: 92.57%
```

7 Implementing Cosine Similarity

The Cosine Similarity algorithm is implemented to retrieve the counselling responses based on similarity scores. Initially the algorithm is implemented by utilizing the fine-tuned BERT Algorithm. So, the Model is loaded as shown below,

```
#Implementing Cosine Similarity Algorithm
# Load the fine-tuned model and tokenizer
finetune_model = BertForSequenceClassification.from_pretrained("/content/drive/MyDrive/MasterThesisChatBot/EMP_CB_finetune")
tokenizer = BertTokenizer.from_pretrained("/content/drive/MyDrive/MasterThesisChatBot/EMP_CB_finetune")

prepdata = pd.read_excel("/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/prepdata.xlsx")

# Define a reverse mapping of encoded emotions to labels
encoded_to_emotions = {
    38: "Overwhelmed",
    2: "Anxious",
    26: "Fearful",
    45: "Seeking guidance",
    44: "Seeking advice",
    4: "Appreciative",
    22: "Down",
    27: "Frustrated",
    7: "Apprehensive",
    30: "Grateful",
    59: "Worried",
    40: "Pressured",
    58: "Valued",
    47: "Stressed",
    32: "Hesitant",
    21: "Dissatisfied",
    8: "Bored",
    53: "Unfulfilled",
    0: "Afraid",
    10: "Comforted",
    51: "Unappreciated",
    57: "Unsatisfied",
    12: "Concerned",
    41: "Proactive",
    33: "Hopeful",
    42: "Reassured",
    36: "Overloaded",
    15: "Demotivated",
    49: "Struggling",
    18: "Disappointed",
    19: "Disheartened",
    50: "Suspicious",
    11: "Concern",
    31: "Gratitude",
    29: "Frustration",
```

```
def get_response(input_text):
    # Tokenize input text
    input_ids = tokenizer.encode(input_text, add_special_tokens=True, return_tensors="pt")

    # Get model's prediction
    with torch.no_grad():
        logits = model(input_ids).logits
        predicted_label = torch.argmax(logits, dim=1).item()

    # Check if the predicted label is valid
    if predicted_label in encoded_to_emotions:
        predicted_emotion = encoded_to_emotions[predicted_label]

        # Calculate cosine similarity for unique responses and find the most similar emotion response
        most_similar_response = None
        max_similarity = -1

        for index, row in prepdata.iterrows():
            if row['emotions_encoded'] == predicted_label:
                response = row['answerText']

                response_ids = tokenizer.encode(response, add_special_tokens=True, return_tensors="pt")
                similarity = cosine_similarity(logits.detach().numpy(), model(response_ids).logits.detach().numpy()).item()

                if similarity > max_similarity:
                    max_similarity = similarity
                    most_similar_response = response

        return most_similar_response
    else:
        return "I'm not sure how to respond."
```

```
def chatbot_main():
    print("Chatbot: Hello! How can I help you?")
    while True:
        user_input = input("You: ")
        if user_input.lower() in ["exit", "quit", "bye"]:
            print("Chatbot: Goodbye!")
            break

        response = get_response(user_input)

        if response:
            print(f"Chatbot: {response}")
        else:
            print("Chatbot: I'm not sure how to respond to that.")

if __name__ == "__main__":
    chatbot_main()
```

```
Chatbot: Hello! How can I help you?
You: i am feeling overwhelmed
Chatbot: Seeking support is a positive step. Counseling can provide you with tools to navigate the challenges of deal:
You: thank you for the advice. so what should i do?
Chatbot: I appreciate you reaching out. Absenteeism can have various underlying causes, and it's important to explore
You: exit
Chatbot: Goodbye!
```

8 SVM for Model Evaluation

Support Vector Machine (SVM) is used for classifying the texts and to compare the results with the developed research study. So, initially the libraries are installed as below,

```
pip install numpy pandas
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 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) (2023.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

```
import numpy as np # linear algebra
import pandas as pd
```

```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.6)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.0)
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Load the dataset from Excel
data = pd.read_excel("/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/emp_burnout.xlsx")
data
```

	qnID	questionText	answerText	emotions
0	1	Hey, I've been feeling really overwhelmed and ...	I'm glad you reached out. It's important to ad...	Overwhelmed
1	2	Well, the workload has been increasing, and th...	I understand how challenging that can be. It s...	Anxious
2	3	Not yet. I'm afraid they'll see it as a weakne...	It's important to remember that asking for sup...	Fearful
3	4	That makes sense. I guess I just need to find ...	Absolutely. Start by scheduling a meeting with...	Seeking guidance
4	5	Thank you for the advice. I'll try to have tha...	Certainly. It's important to prioritize self-c...	Seeking advice
...	...	...	...	...
291	292	I've been feeling incredibly stressed and over...	Thank you for sharing your concerns. Dealing w...	Stressed
292	293	I have multiple projects with deadlines that s...	I'm sorry to hear that you're going through th...	Anxious
293	294	I haven't talked to anyone at work about it. I...	Seeking support is a positive step. Counseling...	Worried
294	295	Thank you for the suggestion. I'll consider co...	Certainly. While considering counseling, there...	Appreciative
295	296	Thank you for those suggestions. I'll make an ...	You're welcome. Remember, seeking support is a...	Appreciative

296 rows × 4 columns

Data preprocessing and augmentation steps are carried out to implement the model as shown below,

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

# Clean and preprocess text data
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespaces
    return text

data['questionText'] = data['questionText'].apply(preprocess_text)

# Lemmatize
def lemmatize(text):
    lemmatizer = WordNetLemmatizer()
    words = word_tokenize(text)  # Tokenize the text
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(lemmatized_words)

data['questionText'] = data['questionText'].apply(lemmatize)

# Remove stopwords
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    filtered_words = [word for word in text.split() if word not in stop_words]
    return ' '.join(filtered_words)

data['questionText'] = data['questionText'].apply(remove_stopwords)

# Train Word2Vec model on your preprocessed data
sentences = [word_tokenize(text) for text in data['questionText']]
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)
```

```python
# Save the preprocessed dataset
data.to_excel('/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/prepnewnostem.xlsx', index=False)
print(data)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
     qnID                                       questionText  \
0       1  hey ive feeling really overwhelmed stressed la...
1       2  well workload ha increasing tight deadline mee...
2       3  yet im afraid theyll see weakness think cant h...
3       4  make sense guess need find right way approach ...
4       5  thank advice ill try conversation soon aside a...
..    ...                                                ...
291   292  ive feeling incredibly stressed overwhelmed im...
292   293  multiple project deadline seem almost impossib...
293   294  havent talked anyone work im concerned admitti...
294   295  thank suggestion ill consider counseling explo...
295   296  thank suggestion ill make effort implement fin...

                                            answerText         emotions
0    I'm glad you reached out. It's important to ad...      Overwhelmed
1    I understand how challenging that can be. It s...          Anxious
2    It's important to remember that asking for sup...          Fearful
3    Absolutely. Start by scheduling a meeting with...  Seeking guidance
4    Certainly. It's important to prioritize self-c...   Seeking advice
..                                                 ...              ...
291  Thank you for sharing your concerns. Dealing w...         Stressed
292  I'm sorry to hear that you're going through th...          Anxious
293  Seeking support is a positive step. Counseling...          Worried
294  Certainly. While considering counseling, there...     Appreciative
295  You're welcome. Remember, seeking support is a...     Appreciative

[296 rows x 4 columns]
```

```
pip install googletrans==4.0.0-rc1
```

```
Collecting googletrans==4.0.0-rc1
  Downloading googletrans-4.0.0rc1.tar.gz (20 kB)
  Preparing metadata (setup.py) ... done
Collecting httpx==0.13.3 (from googletrans==4.0.0-rc1)
  Downloading httpx-0.13.3-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.1/55.1 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (2023.7.22)
Collecting hstspreload (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading hstspreload-2023.1.1-py3-none-any.whl (1.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 7.6 MB/s eta 0:00:00
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from httpx==0.13.3->googletrans==4.0.0-rc1) (1.3.0)
Collecting chardet==3.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.4/133.4 kB 10.8 MB/s eta 0:00:00
Collecting idna==2.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 6.6 MB/s eta 0:00:00
Collecting rfc3986<2,>=1.3 (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading rfc3986-1.5.0-py2.py3-none-any.whl (31 kB)
Collecting httpcore==0.9.* (from httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading httpcore-0.9.1-py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 4.9 MB/s eta 0:00:00
Collecting h11<0.10,>=0.8 (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1)
  Downloading h11-0.9.0-py2.py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.6/53.6 kB 6.2 MB/s eta 0:00:00
Collecting h2==3.* (from httpcore==0.9.*->httpx==0.13.3->googletrans==4.0.0-rc1)
```

```python
from nltk.corpus import wordnet
from googletrans import Translator

# Synonym Replacement
def synonym_replacement(text):
    words = word_tokenize(text)
    new_words = []
    for word in words:
        synonyms = wordnet.synsets(word)
        if synonyms:
            synonym = synonyms[0].lemmas()[0].name()
            new_words.append(synonym)
        else:
            new_words.append(word)
    return ' '.join(new_words)

def rephrase_question(text):
    tokens = word_tokenize(text)
    rephrased_tokens = [synonym_replacement(token) for token in tokens]
    return ' '.join(rephrased_tokens)

# Dialogue combination
def dialogue_combination(text, num_samples=1):
    augmented_data = []
    for _ in range(num_samples):
        indexes = random.sample(range(len(data)), 2)
        question1, answer1 = data.loc[indexes[0]]
        question2, answer2 = data.loc[indexes[1]]
        new_question = f"{question1} {question2}"
        new_answer = f"{answer1} {answer2}"
        augmented_data.append((new_question, new_answer))
    return augmented_data
```

```python
# Paraphrasing using Google Translate (English to Spanish and back to English)
def paraphrasing(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='es')
    paraphrased = translator.translate(translation.text, src='es', dest='en')
    return paraphrased.text

# Back-translation using Google Translate (English to French and back to English)
def back_translation(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='fr')
    back_translated = translator.translate(translation.text, src='fr', dest='en')
    return back_translated.text

# Augment the dataframe using data augmentation techniques
augmented_data = []
```

```python
for index, row in data.iterrows():
    question = row['questionText']
    answer = row['answerText']
    emotion = row['emotions']

    # Original data
    augmented_data.append({'questionText': question, 'emotions': emotion, 'answerText': answer})

    # Synonym Replacement
    augmented_data.append({'questionText': synonym_replacement(question), 'emotions': emotion, 'answerText': answer})

    # rephrase_question
    augmented_data.append({'questionText': rephrase_question(question), 'emotions': emotion, 'answerText': answer})

    # Paraphrasing
    augmented_data.append({'questionText': paraphrasing(question), 'emotions': emotion, 'answerText': answer})

    # Back-translation
    augmented_data.append({'questionText': back_translation(question), 'emotions': emotion, 'answerText': answer})

# Create augmented dataframe
augmented_df = pd.DataFrame(augmented_data)

# Paraphrasing using Google Translate (English to Spanish and back to English)
def paraphrasing(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='es')
    paraphrased = translator.translate(translation.text, src='es', dest='en')
    return paraphrased.text

# Back-translation using Google Translate (English to French and back to English)
def back_translation(text):
    translator = Translator()
    translation = translator.translate(text, src='en', dest='fr')
    back_translated = translator.translate(translation.text, src='fr', dest='en')
    return back_translated.text

# Augment the dataframe using data augmentation techniques
augmented_data = []

for index, row in data.iterrows():
    question = row['questionText']
    answer = row['answerText']
    emotion = row['emotions']

    # Original data
    augmented_data.append({'questionText': question, 'emotions': emotion, 'answerText': answer})

    # Synonym Replacement
    augmented_data.append({'questionText': synonym_replacement(question), 'emotions': emotion, 'answerText': answer})

    # rephrase_question
    augmented_data.append({'questionText': rephrase_question(question), 'emotions': emotion, 'answerText': answer})

    # Paraphrasing
    augmented_data.append({'questionText': paraphrasing(question), 'emotions': emotion, 'answerText': answer})

    # Back-translation
    augmented_data.append({'questionText': back_translation(question), 'emotions': emotion, 'answerText': answer})
```

```
# Display augmented dataframe
print(augmented_df)
```

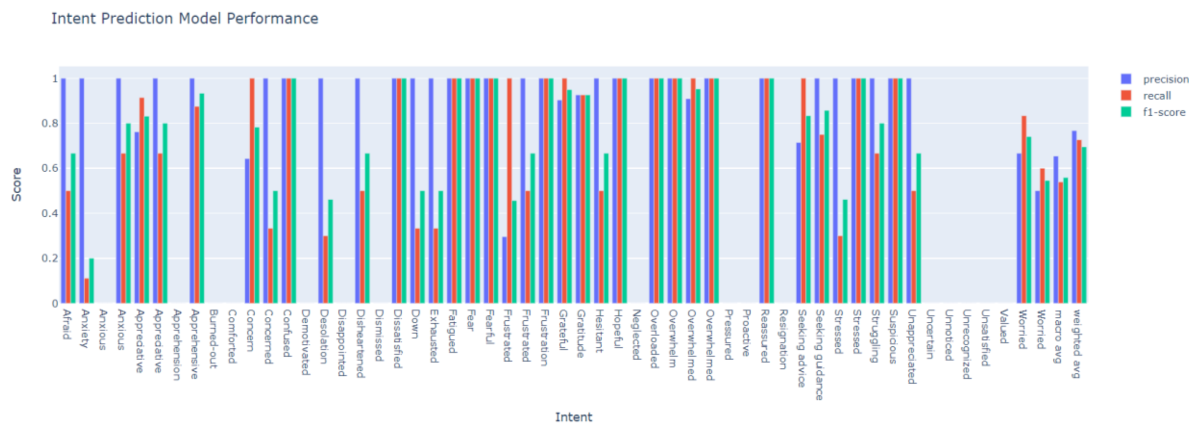
```
                                           questionText       emotions  \
0     hey ive feeling really overwhelmed stressed la...    Overwhelmed
1     hey ive feeling truly overwhelm stress recentl...    Overwhelmed
2     hey ive feeling truly overwhelm stress recentl...    Overwhelmed
3     Hey, I feel very overwhelmed stressed lately, ...    Overwhelmed
4     Hey, I feel really overwhelmed in recent times...    Overwhelmed
...                                                 ...            ...
1475  thank suggestion ill make effort implement fin...  Appreciative
1476  thank suggestion ailment brand attempt impleme...  Appreciative
1477  thank suggestion ailment brand attempt impleme...  Appreciative
1478  Appreciate suggestion.I will make the effort i...  Appreciative
1479  thank you suggestion badly making efforts to i...  Appreciative

                                             answerText
0     I'm glad you reached out. It's important to ad...
1     I'm glad you reached out. It's important to ad...
2     I'm glad you reached out. It's important to ad...
3     I'm glad you reached out. It's important to ad...
4     I'm glad you reached out. It's important to ad...
...                                                 ...
1475  You're welcome. Remember, seeking support is a...
1476  You're welcome. Remember, seeking support is a...
1477  You're welcome. Remember, seeking support is a...
1478  You're welcome. Remember, seeking support is a...
1479  You're welcome. Remember, seeking support is a...

[1480 rows x 3 columns]
```

Intent Prediction Model

```
import joblib
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import plotly.graph_objs as go
from sklearn.metrics import accuracy_score

# Split the dataset into training and testing sets
X = augmented_df['questionText']
y = augmented_df['emotions']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train a Support Vector Machine (SVM) classifier
model = SVC()
model.fit(X_train_vec, y_train)

# Save the trained model and vectorizer to specified paths
model_path = '/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/trained_model.pkl'
vectorizer_path = '/content/drive/MyDrive/MasterThesisChatBot/EMP_CB/vectorizer.pkl'

joblib.dump(model, model_path)
joblib.dump(vectorizer, vectorizer_path)

# Predict intents for the testing set
y_pred = model.predict(X_test_vec)

# Calculate model accuracy
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)
```

```python
# Evaluate the model's performance
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)

# Convert float values in the report to dictionaries
report = {label: {metric: report[label][metric] for metric in report[label]} for label in report if isinstance(report[label], dict)}

# Extract evaluation metrics
labels = list(report.keys())
evaluation_metrics = ['precision', 'recall', 'f1-score']
metric_scores = {metric: [report[label][metric] for label in labels if label in report] for metric in evaluation_metrics}

# Visualize the model's performance using a Plotly bar plot
fig = go.Figure()
for metric in evaluation_metrics:
    fig.add_trace(go.Bar(name=metric, x=labels, y=metric_scores[metric]))

fig.update_layout(title='Intent Prediction Model Performance',
                  xaxis_title='Intent',
                  yaxis_title='Score',
                  barmode='group')

fig.show()
```

```
Model Accuracy: 0.7263513513513513
```




```python
from sklearn.metrics import classification_report
all_labels = sorted(np.unique(np.concatenate((y_test, y_pred))))
print(classification_report(y_test, y_pred, labels=all_labels, target_names=all_labels))
```

```
               precision    recall  f1-score   support

       Afraid       1.00      0.50      0.67         2
      Anxiety       1.00      0.11      0.20         9
      Anxious       0.00      0.00      0.00         1
     Anxious        1.00      0.67      0.80         3
  Appreciative      0.76      0.91      0.83        35
 Appreciative       1.00      0.67      0.80        12
  Apprehension      0.00      0.00      0.00         2
  Apprehensive      1.00      0.88      0.93         8
   Burned-out       0.00      0.00      0.00         1
     Comforted      0.00      0.00      0.00         4
       Concern      0.64      1.00      0.78         9
     Concerned      1.00      0.33      0.50         3
      Confused      1.00      1.00      1.00         1
   Demotivated      0.00      0.00      0.00         1
    Desolation      1.00      0.30      0.46        10
  Disappointed      0.00      0.00      0.00         1
  Disheartened      1.00      0.50      0.67         2
     Dismissed      0.00      0.00      0.00         2
  Dissatisfied      1.00      1.00      1.00         4
          Down      1.00      0.33      0.50         3
    Exhausted       1.00      0.33      0.50         3
     Fatigued       1.00      1.00      1.00         1
          Fear      1.00      1.00      1.00         1
       Fearful      1.00      1.00      1.00         3
    Frustrated      0.30      1.00      0.46        21
   Frustrated       1.00      0.50      0.67         2
   Frustration      1.00      1.00      1.00         3
```

```
    accuracy                           0.73       296
   macro avg       0.65      0.54      0.56       296
weighted avg       0.77      0.73      0.70       296
```