Configuration Manual

MSc Research Project
Data Analytics

Syed Ebrahim Abdul Kareem
Student ID: x20232616

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Syed Ebrahim Abdul Kareem…………………………………………………
Student ID:	X20232616…………………………………………………………………………….……
Programme:	Data Analytics……………………………… **Year:** 2022……………..
Module:	MSc Research Project……………………………………………………….………
Lecturer:	Vladimir Milosavljevic………………………………………………………….………
Submission Due Date:	01/02/2023…………………………………………………………………………….………
Project Title:	Leveraging Transfer Learning Techniques for Homophobia and Transphobia Detection
Word Count:	768…………………………… **Page Count:** 13…………………………….…….………

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: Syed Ebrahim Abdul Kareem ……………………………………………

Date: 01/02/2023………………………………………………………………………

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

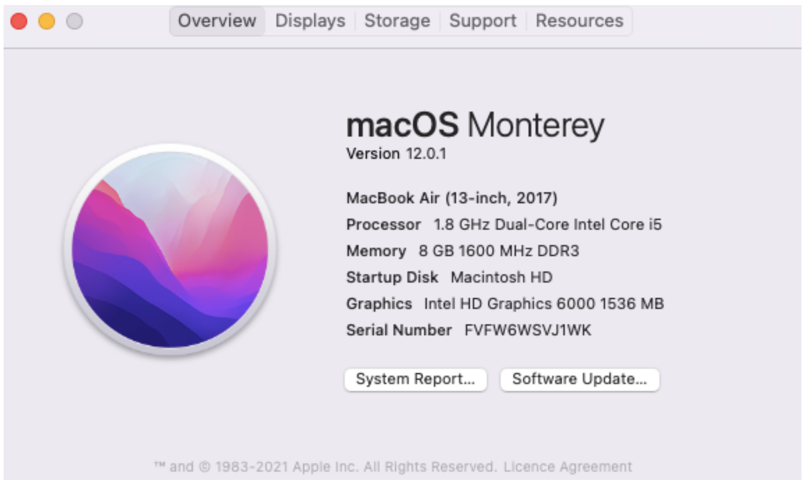
Syed Ebrahim Abdul Kareem
x20232616

1 Introduction

This document provides information about the hardware and software components used to build the homophobia and transphobia detection system. The steps outlined in this manual can be followed to run the code and reproduce the results.

2 Hardware and Software Configuration

The technical specifications of host device used to implement this research project is shown in Figure 1



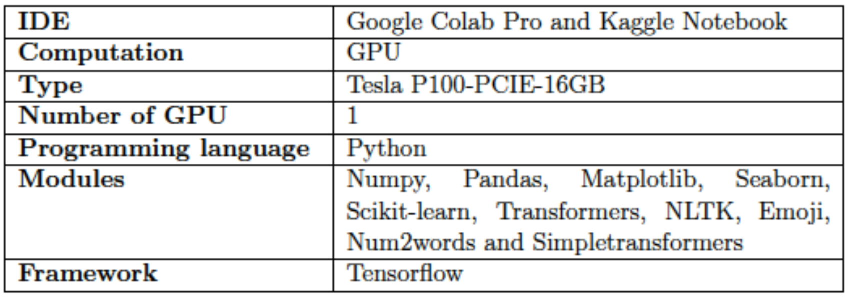
macOS Monterey
Version 12.0.1

MacBook Air (13-inch, 2017)

Processor	1.8 GHz Dual-Core Intel Core i5
Memory	8 GB 1600 MHz DDR3
Startup Disk	Macintosh HD
Graphics	Intel HD Graphics 6000 1536 MB
Serial Number	FVFW6WSVJ1WK

Figure 1: Technical Configuration

Python 3.7 is used for implementation. All the frameworks and libraries required are given below in Figure 2.



IDE	Google Colab Pro and Kaggle Notebook
Computation	GPU
Type	Tesla P100-PCIE-16GB
Number of GPU	1
Programming language	Python
Modules	Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Transformers, NLTK, Emoji, Num2words and Simpletransformers
Framework	Tensorflow

Figure 2: Setup Configuration

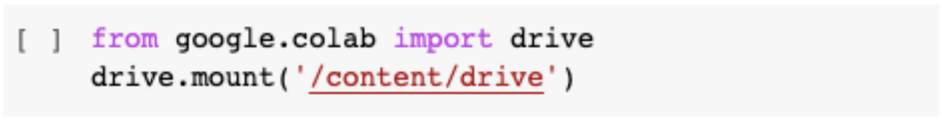
3 Dataset

The dataset used in this research was compiled by (Chakravarthi et al., n.d.). Dataset was directly downloaded from https://codalab.lisn.upsaclay.fr/competitions/5310. The dataset was already splitted into train, validation and test data. The data is uploaded to Google drive and Kaggle account.

4 Implementation on Models

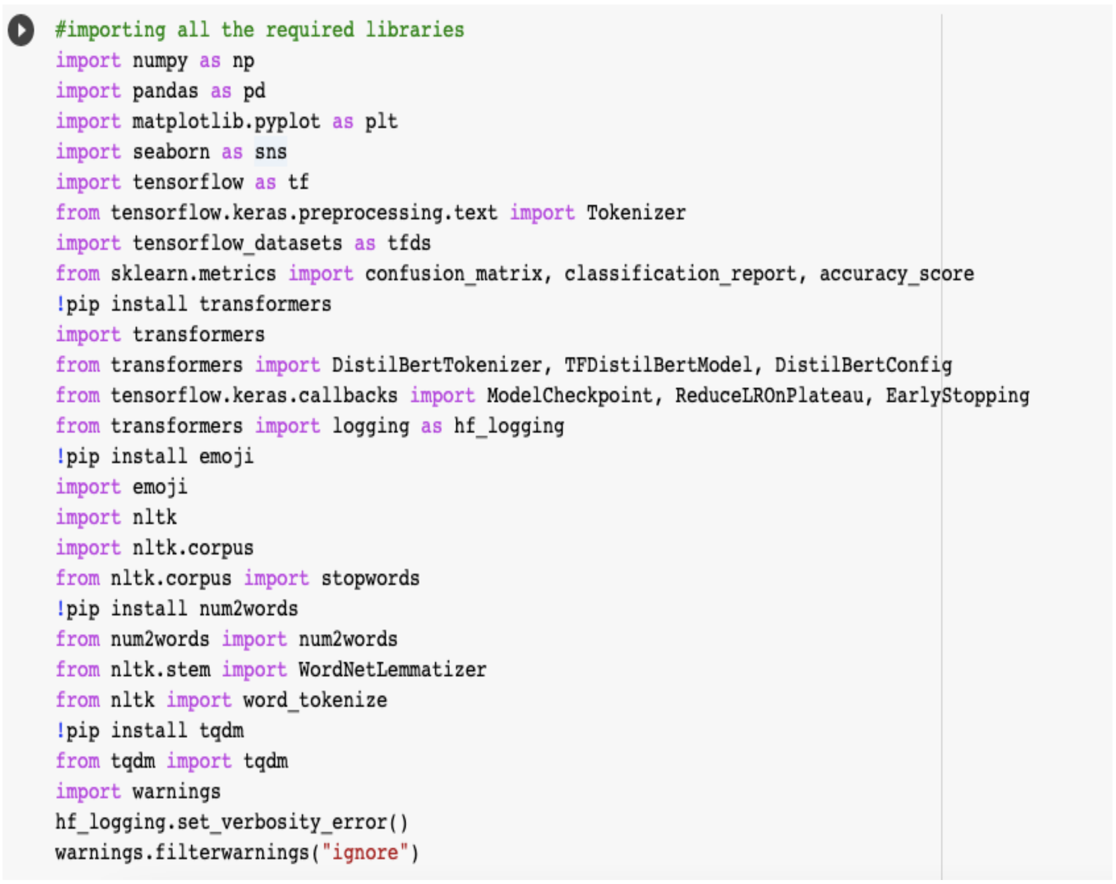
4.1 DistilBERT

Google drive is mounted on the current session of Colab Notebook to read the dataset using the code given in Figure 3.



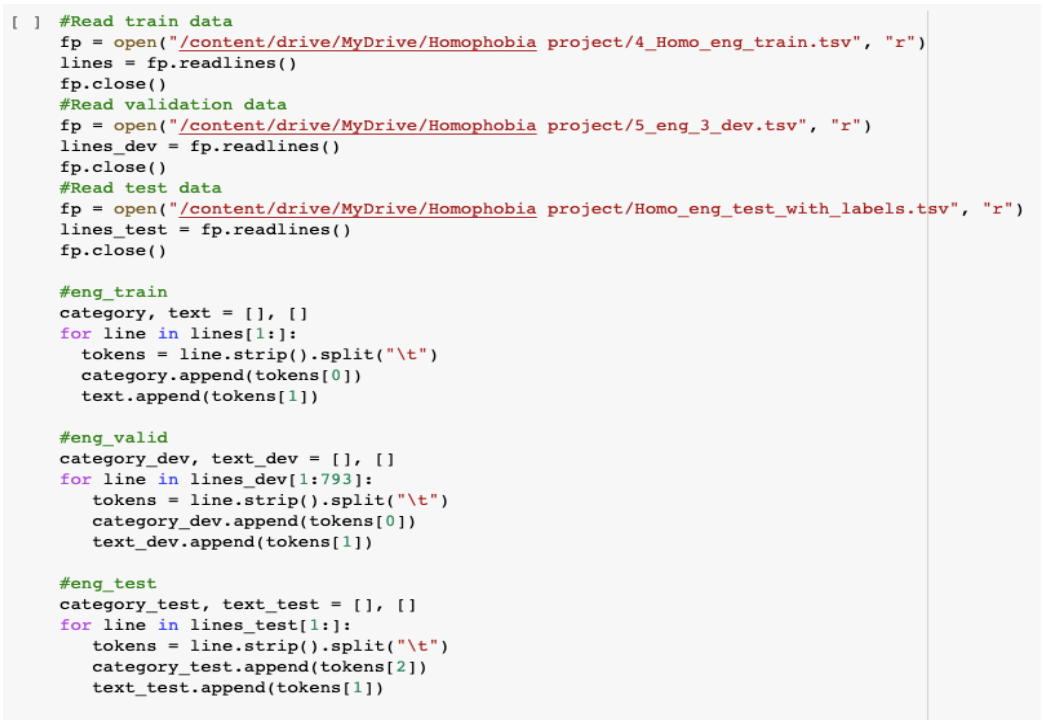
```
from google.colab import drive
drive.mount('/content/drive')
```

Figure 3: Mount google drive



```
#importing all the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
import tensorflow_datasets as tfds
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
!pip install transformers
import transformers
from transformers import DistilBertTokenizer, TFDistilBertModel, DistilBertConfig
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from transformers import logging as hf_logging
!pip install emoji
import emoji
import nltk
import nltk.corpus
from nltk.corpus import stopwords
!pip install num2words
from num2words import num2words
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
!pip install tqdm
from tqdm import tqdm
import warnings
hf_logging.set_verbosity_error()
warnings.filterwarnings("ignore")
```

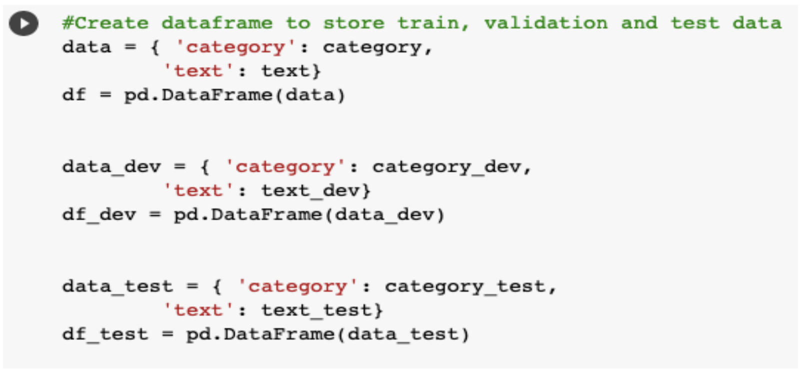
Figure 4: Importing all the required libraries



```
[ ] #Read train data
    fp = open("/content/drive/MyDrive/Homophobia project/4_Homo_eng_train.tsv", "r")
    lines = fp.readlines()
    fp.close()
    #Read validation data
    fp = open("/content/drive/MyDrive/Homophobia project/5_eng_3_dev.tsv", "r")
    lines_dev = fp.readlines()
    fp.close()
    #Read test data
    fp = open("/content/drive/MyDrive/Homophobia project/Homo_eng_test_with_labels.tsv", "r")
    lines_test = fp.readlines()
    fp.close()

    #eng_train
    category, text = [], []
    for line in lines[1:]:
      tokens = line.strip().split("\t")
      category.append(tokens[0])
      text.append(tokens[1])

    #eng_valid
    category_dev, text_dev = [], []
    for line in lines_dev[1:793]:
       tokens = line.strip().split("\t")
       category_dev.append(tokens[0])
       text_dev.append(tokens[1])

    #eng_test
    category_test, text_test = [], []
    for line in lines_test[1:]:
       tokens = line.strip().split("\t")
       category_test.append(tokens[2])
       text_test.append(tokens[1])
```

Figure 5: Read the data files and extract text and labels

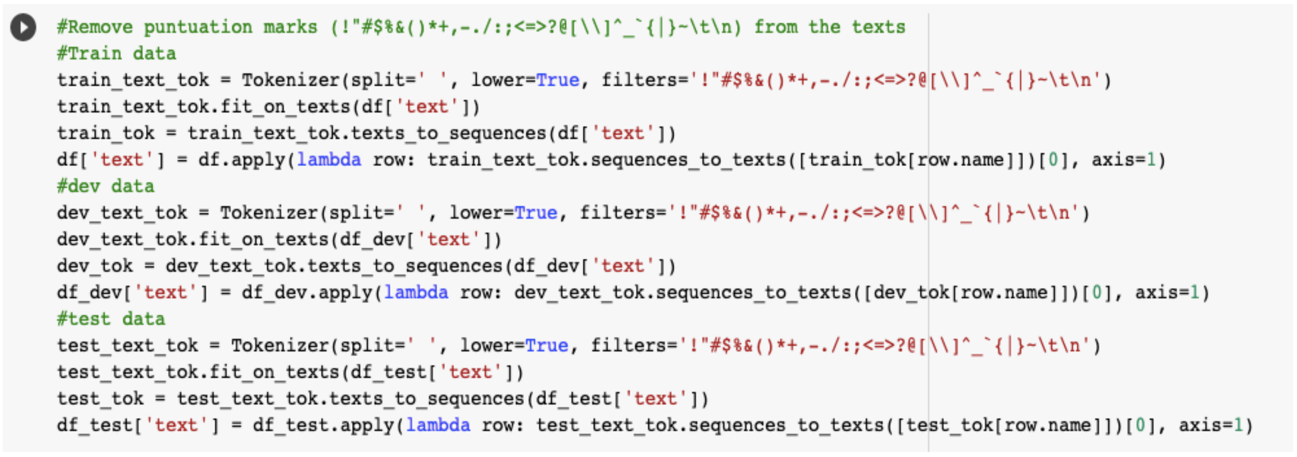
Figure 6 shows the code used to store train, validation and test data in pandas dataframe.



```
#Create dataframe to store train, validation and test data
data = { 'category': category,
        'text': text}
df = pd.DataFrame(data)


data_dev = { 'category': category_dev,
            'text': text_dev}
df_dev = pd.DataFrame(data_dev)


data_test = { 'category': category_test,
             'text': text_test}
df_test = pd.DataFrame(data_test)
```

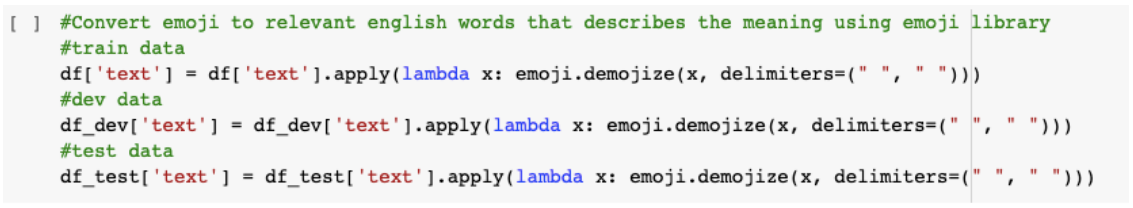
Figure 6: Create Pandas Dataframe to store the data

The data pre-processing steps are given from Figure 7 to 11.



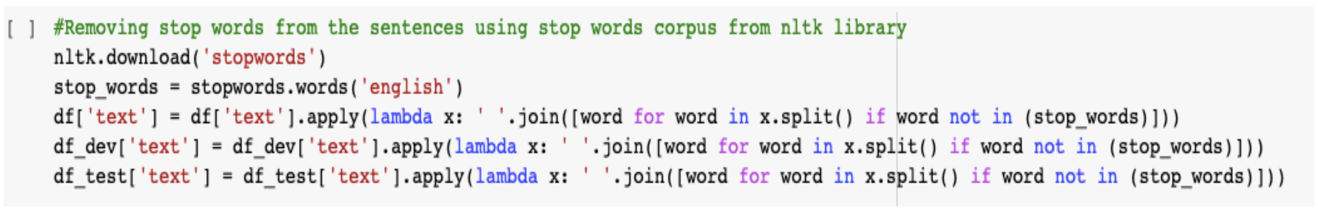
```
#Remove puntuation marks (!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n) from the texts
#Train data
train_text_tok = Tokenizer(split=' ', lower=True, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
train_text_tok.fit_on_texts(df['text'])
train_tok = train_text_tok.texts_to_sequences(df['text'])
df['text'] = df.apply(lambda row: train_text_tok.sequences_to_texts([train_tok[row.name]])[0], axis=1)
#dev data
dev_text_tok = Tokenizer(split=' ', lower=True, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
dev_text_tok.fit_on_texts(df_dev['text'])
dev_tok = dev_text_tok.texts_to_sequences(df_dev['text'])
df_dev['text'] = df_dev.apply(lambda row: dev_text_tok.sequences_to_texts([dev_tok[row.name]])[0], axis=1)
#test data
test_text_tok = Tokenizer(split=' ', lower=True, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
test_text_tok.fit_on_texts(df_test['text'])
test_tok = test_text_tok.texts_to_sequences(df_test['text'])
df_test['text'] = df_test.apply(lambda row: test_text_tok.sequences_to_texts([test_tok[row.name]])[0], axis=1)
```

Figure 7: Punctuation removal



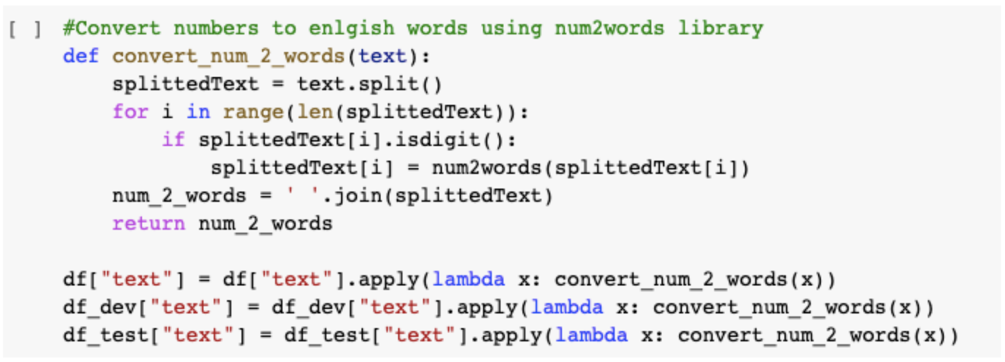
```python
#Convert emoji to relevant english words that describes the meaning using emoji library
#train data
df['text'] = df['text'].apply(lambda x: emoji.demojize(x, delimiters=(" ", " ")))
#dev data
df_dev['text'] = df_dev['text'].apply(lambda x: emoji.demojize(x, delimiters=(" ", " ")))
#test data
df_test['text'] = df_test['text'].apply(lambda x: emoji.demojize(x, delimiters=(" ", " ")))
```

Figure 8: Emoji conversion



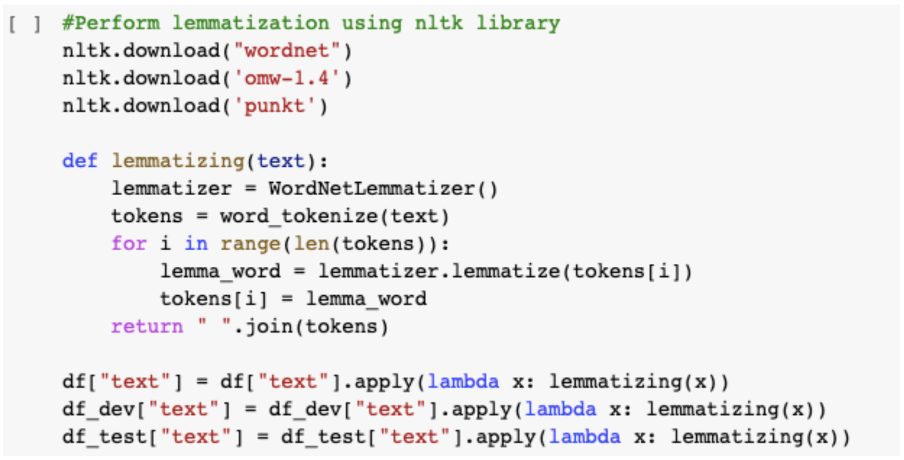
```python
#Removing stop words from the sentences using stop words corpus from nltk library
nltk.download('stopwords')
stop_words = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
df_dev['text'] = df_dev['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
df_test['text'] = df_test['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
```

Figure 9: Stop words removal



```python
#Convert numbers to enlgish words using num2words library
def convert_num_2_words(text):
    splittedText = text.split()
    for i in range(len(splittedText)):
        if splittedText[i].isdigit():
            splittedText[i] = num2words(splittedText[i])
    num_2_words = ' '.join(splittedText)
    return num_2_words

df["text"] = df["text"].apply(lambda x: convert_num_2_words(x))
df_dev["text"] = df_dev["text"].apply(lambda x: convert_num_2_words(x))
df_test["text"] = df_test["text"].apply(lambda x: convert_num_2_words(x))
```

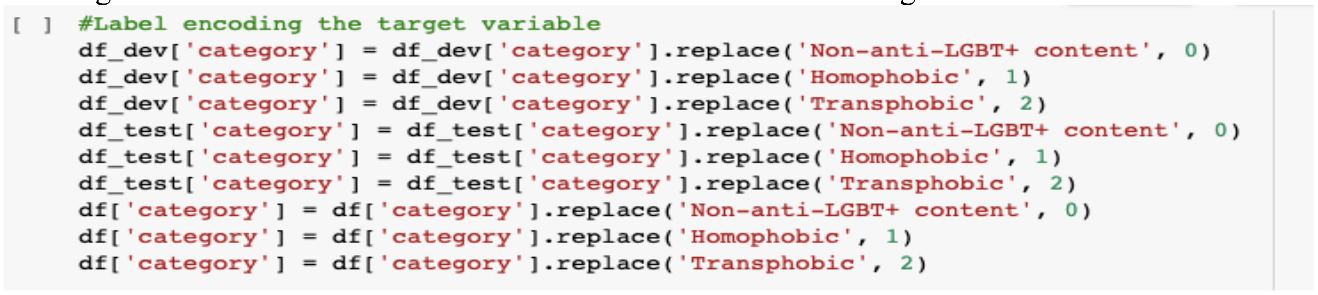
Figure 10: Convert numbers to words



```python
#Perform lemmatization using nltk library
nltk.download("wordnet")
nltk.download('omw-1.4')
nltk.download('punkt')

def lemmatizing(text):
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text)
    for i in range(len(tokens)):
        lemma_word = lemmatizer.lemmatize(tokens[i])
        tokens[i] = lemma_word
    return " ".join(tokens)

df["text"] = df["text"].apply(lambda x: lemmatizing(x))
df_dev["text"] = df_dev["text"].apply(lambda x: lemmatizing(x))
df_test["text"] = df_test["text"].apply(lambda x: lemmatizing(x))
```

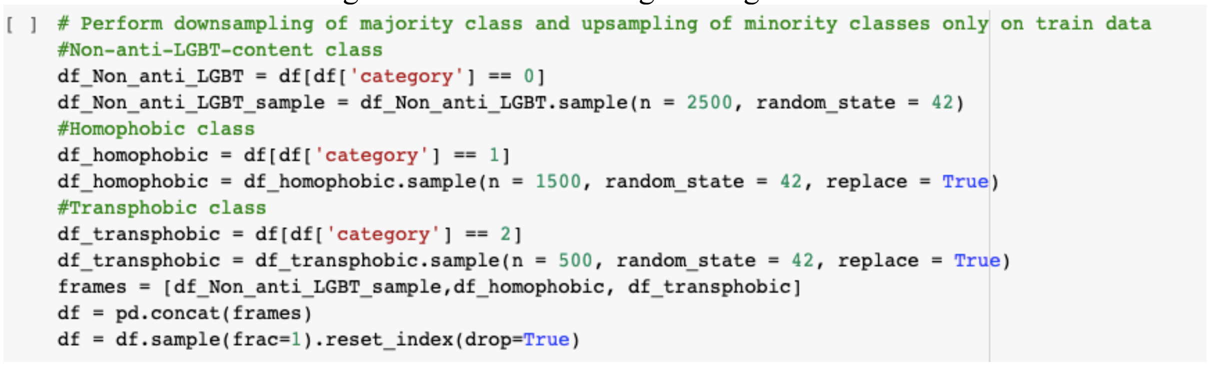
Figure 11: Lemmatization

The target variable is label encoded into 3 classes. This is shown in Figure 12.



```python
#Label encoding the target variable
df_dev['category'] = df_dev['category'].replace('Non-anti-LGBT+ content', 0)
df_dev['category'] = df_dev['category'].replace('Homophobic', 1)
df_dev['category'] = df_dev['category'].replace('Transphobic', 2)
df_test['category'] = df_test['category'].replace('Non-anti-LGBT+ content', 0)
df_test['category'] = df_test['category'].replace('Homophobic', 1)
df_test['category'] = df_test['category'].replace('Transphobic', 2)
df['category'] = df['category'].replace('Non-anti-LGBT+ content', 0)
df['category'] = df['category'].replace('Homophobic', 1)
df['category'] = df['category'].replace('Transphobic', 2)
```

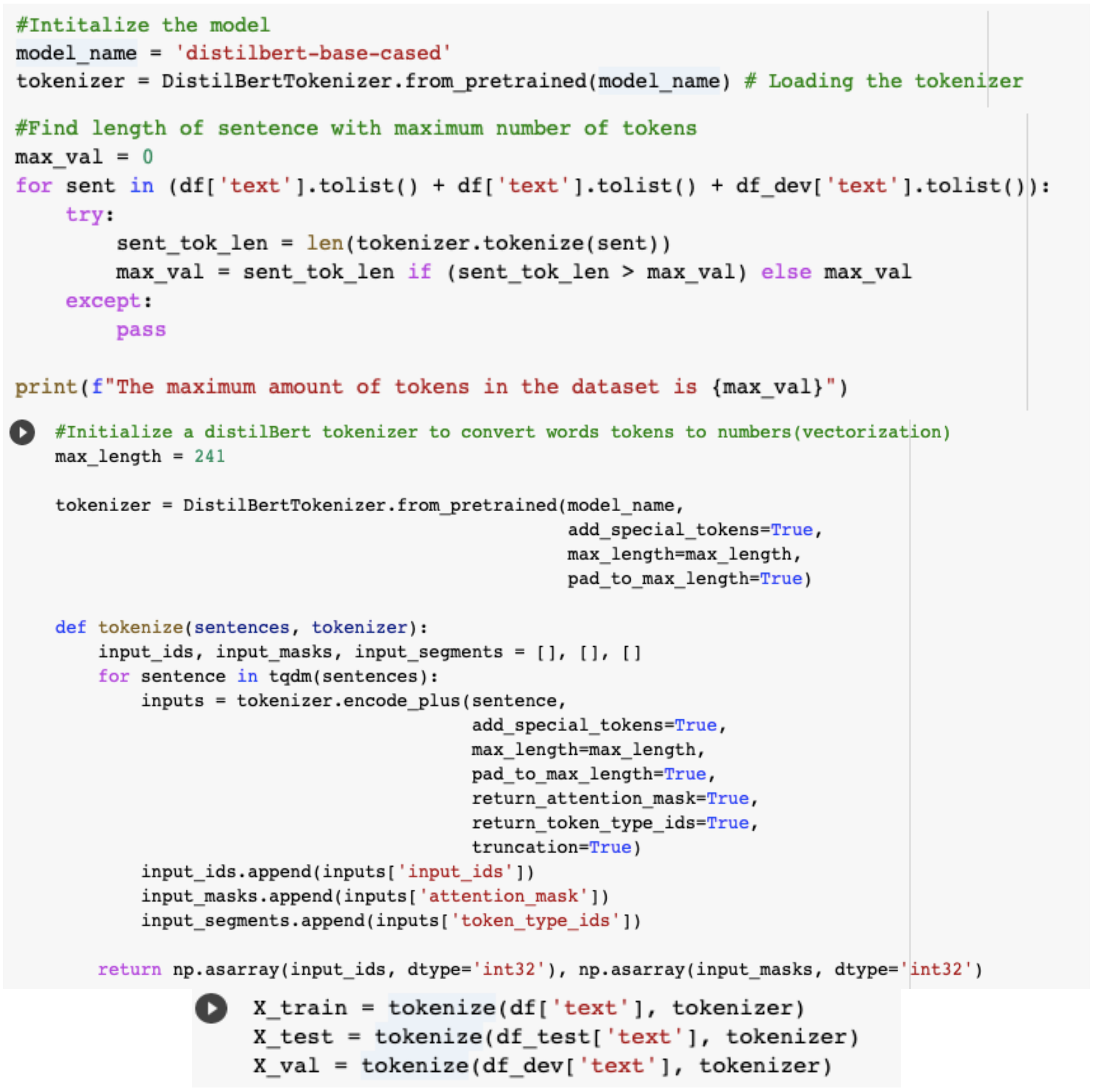
Figure 12: Label encoding the target variable



```
[ ] # Perform downsampling of majority class and upsampling of minority classes only on train data
    #Non-anti-LGBT-content class
    df_Non_anti_LGBT = df[df['category'] == 0]
    df_Non_anti_LGBT_sample = df_Non_anti_LGBT.sample(n = 2500, random_state = 42)
    #Homophobic class
    df_homophobic = df[df['category'] == 1]
    df_homophobic = df_homophobic.sample(n = 1500, random_state = 42, replace = True)
    #Transphobic class
    df_transphobic = df[df['category'] == 2]
    df_transphobic = df_transphobic.sample(n = 500, random_state = 42, replace = True)
    frames = [df_Non_anti_LGBT_sample,df_homophobic, df_transphobic]
    df = pd.concat(frames)
    df = df.sample(frac=1).reset_index(drop=True)
```

Figure 13: Combination of upsampling and downsampling technique

Figure 14 shows the code snippet for Vectorisation. In this step, the text data is converted into vectors.



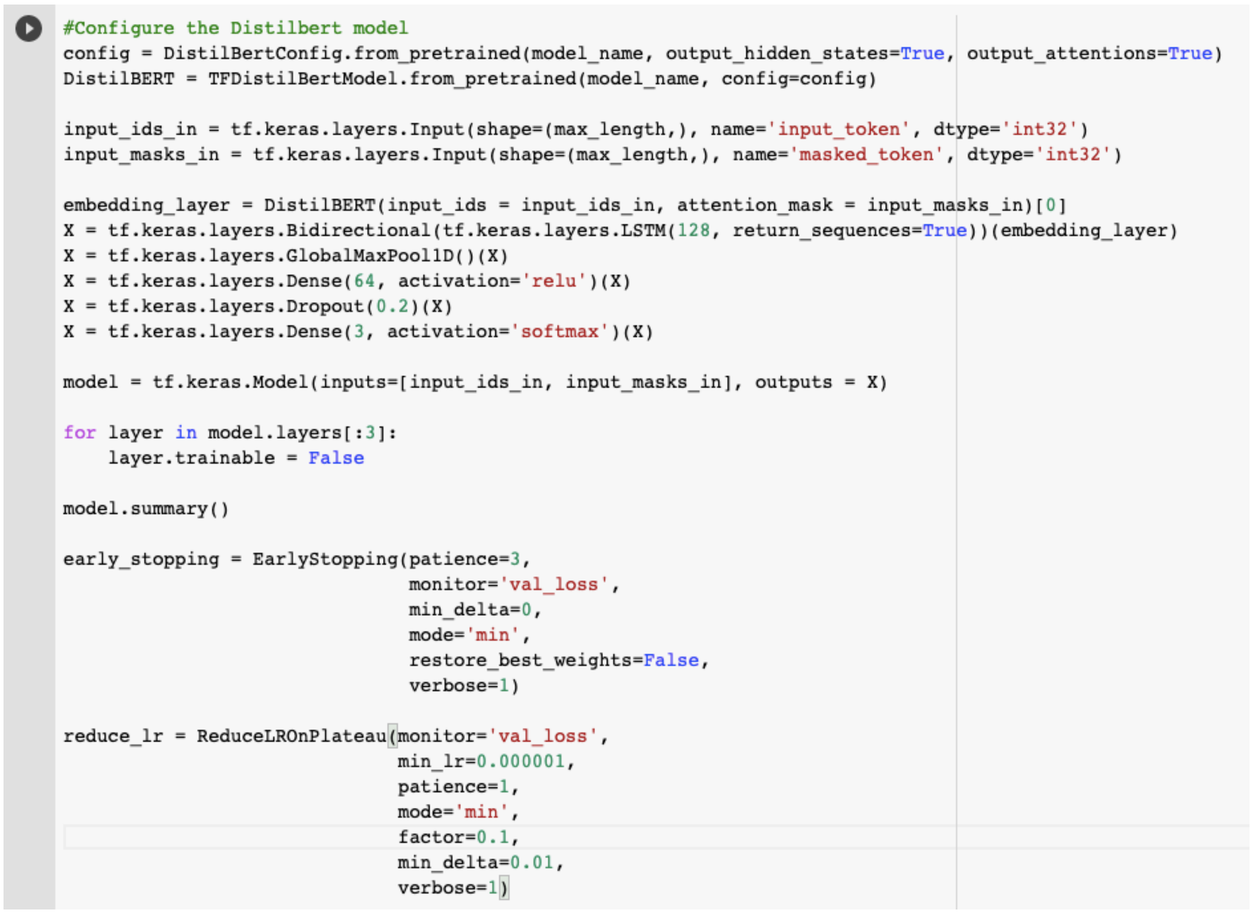
```
#Intitalize the model
model_name = 'distilbert-base-cased'
tokenizer = DistilBertTokenizer.from_pretrained(model_name) # Loading the tokenizer

#Find length of sentence with maximum number of tokens
max_val = 0
for sent in (df['text'].tolist() + df['text'].tolist() + df_dev['text'].tolist()):
    try:
        sent_tok_len = len(tokenizer.tokenize(sent))
        max_val = sent_tok_len if (sent_tok_len > max_val) else max_val
    except:
        pass

print(f"The maximum amount of tokens in the dataset is {max_val}")
```

```
#Initialize a distilBert tokenizer to convert words tokens to numbers(vectorization)
max_length = 241

tokenizer = DistilBertTokenizer.from_pretrained(model_name,
                                                add_special_tokens=True,
                                                max_length=max_length,
                                                pad_to_max_length=True)

def tokenize(sentences, tokenizer):
    input_ids, input_masks, input_segments = [], [], []
    for sentence in tqdm(sentences):
        inputs = tokenizer.encode_plus(sentence,
                                       add_special_tokens=True,
                                       max_length=max_length,
                                       pad_to_max_length=True,
                                       return_attention_mask=True,
                                       return_token_type_ids=True,
                                       truncation=True)
        input_ids.append(inputs['input_ids'])
        input_masks.append(inputs['attention_mask'])
        input_segments.append(inputs['token_type_ids'])

    return np.asarray(input_ids, dtype='int32'), np.asarray(input_masks, dtype='int32')
```

```
X_train = tokenize(df['text'], tokenizer)
X_test = tokenize(df_test['text'], tokenizer)
X_val = tokenize(df_dev['text'], tokenizer)
```

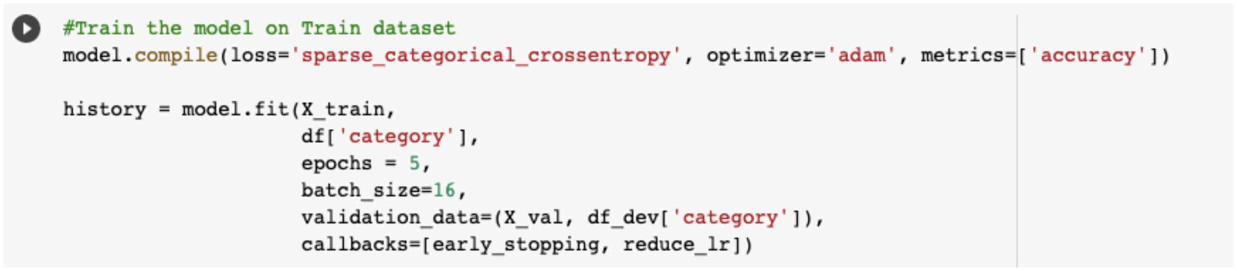
Figure 14: Vectorisation using DistilBERT Tokenizer

Figure 15 shows the code used for configuring the DistilBERT model.



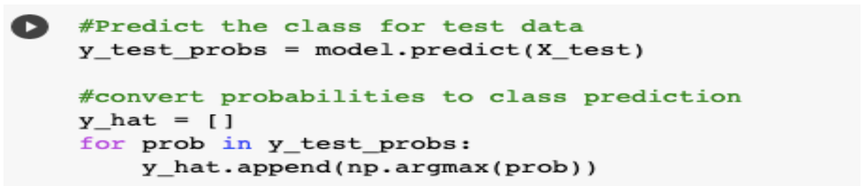
```
#Configure the Distilbert model
config = DistilBertConfig.from_pretrained(model_name, output_hidden_states=True, output_attentions=True)
DistilBERT = TFDistilBertModel.from_pretrained(model_name, config=config)

input_ids_in = tf.keras.layers.Input(shape=(max_length,), name='input_token', dtype='int32')
input_masks_in = tf.keras.layers.Input(shape=(max_length,), name='masked_token', dtype='int32')

embedding_layer = DistilBERT(input_ids = input_ids_in, attention_mask = input_masks_in)[0]
X = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences=True))(embedding_layer)
X = tf.keras.layers.GlobalMaxPool1D()(X)
X = tf.keras.layers.Dense(64, activation='relu')(X)
X = tf.keras.layers.Dropout(0.2)(X)
X = tf.keras.layers.Dense(3, activation='softmax')(X)

model = tf.keras.Model(inputs=[input_ids_in, input_masks_in], outputs = X)

for layer in model.layers[:3]:
    layer.trainable = False

model.summary()

early_stopping = EarlyStopping(patience=3,
                               monitor='val_loss',
                               min_delta=0,
                               mode='min',
                               restore_best_weights=False,
                               verbose=1)

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              min_lr=0.000001,
                              patience=1,
                              mode='min',
                              factor=0.1,
                              min_delta=0.01,
                              verbose=1)
```

Figure 15: Configure the DistilBERT model with hyperparameters



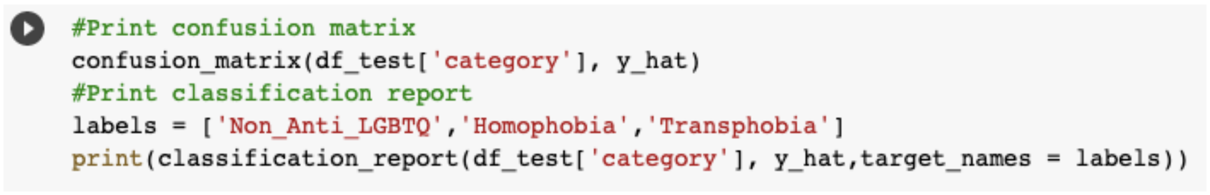
```
#Train the model on Train dataset
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train,
                    df['category'],
                    epochs = 5,
                    batch_size=16,
                    validation_data=(X_val, df_dev['category']),
                    callbacks=[early_stopping, reduce_lr])
```

Figure 16: Model training



```
#Predict the class for test data
y_test_probs = model.predict(X_test)

#convert probabilities to class prediction
y_hat = []
for prob in y_test_probs:
    y_hat.append(np.argmax(prob))
```

Figure 17: Predict the test data



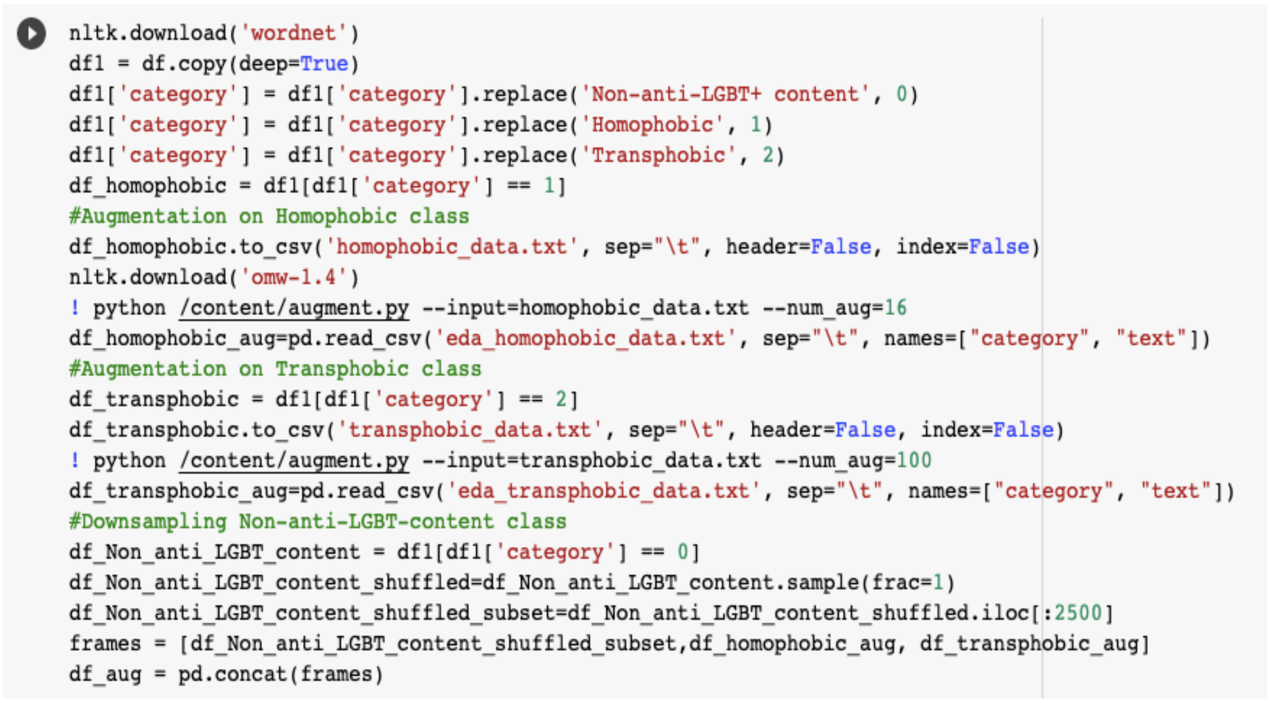
```
#Print confusiion matrix
confusion_matrix(df_test['category'], y_hat)
#Print classification report
labels = ['Non_Anti_LGBTQ','Homophobia','Transphobia']
print(classification_report(df_test['category'], y_hat,target_names = labels))
```

Figure 18: Display Confusion Matrix and Classification Report

4.2 RoBERTa

The data loading and pre-processing steps for RoBERTa models remains the same as DistilBERT model. Only difference is data augmentation and model building. These steps are given below.

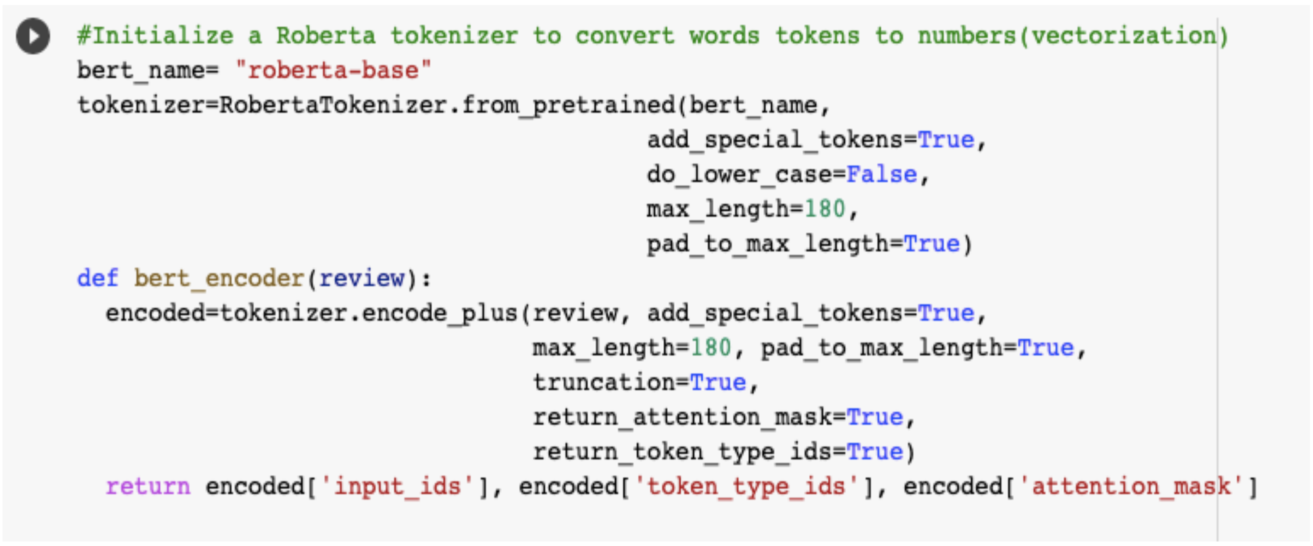
Figure 19 shows the Easy Data Augmentation technique (Wei and Zou, 2019) used to increase the instances for minority classes. To execute this code, two files augment.py and eda.py should be saved in the current working directory. These .py files are attached in the ICT solution code artefacts folder. It can also be found in the following link - https://github.com/jasonwei20/eda_nlp



```
nltk.download('wordnet')
df1 = df.copy(deep=True)
df1['category'] = df1['category'].replace('Non-anti-LGBT+ content', 0)
df1['category'] = df1['category'].replace('Homophobic', 1)
df1['category'] = df1['category'].replace('Transphobic', 2)
df_homophobic = df1[df1['category'] == 1]
#Augmentation on Homophobic class
df_homophobic.to_csv('homophobic_data.txt', sep="\t", header=False, index=False)
nltk.download('omw-1.4')
! python /content/augment.py --input=homophobic_data.txt --num_aug=16
df_homophobic_aug=pd.read_csv('eda_homophobic_data.txt', sep="\t", names=["category", "text"])
#Augmentation on Transphobic class
df_transphobic = df1[df1['category'] == 2]
df_transphobic.to_csv('transphobic_data.txt', sep="\t", header=False, index=False)
! python /content/augment.py --input=transphobic_data.txt --num_aug=100
df_transphobic_aug=pd.read_csv('eda_transphobic_data.txt', sep="\t", names=["category", "text"])
#Downsampling Non-anti-LGBT-content class
df_Non_anti_LGBT_content = df1[df1['category'] == 0]
df_Non_anti_LGBT_content_shuffled=df_Non_anti_LGBT_content.sample(frac=1)
df_Non_anti_LGBT_content_shuffled_subset=df_Non_anti_LGBT_content_shuffled.iloc[:2500]
frames = [df_Non_anti_LGBT_content_shuffled_subset,df_homophobic_aug, df_transphobic_aug]
df_aug = pd.concat(frames)
```

Figure 19: Easy Data Augmentation

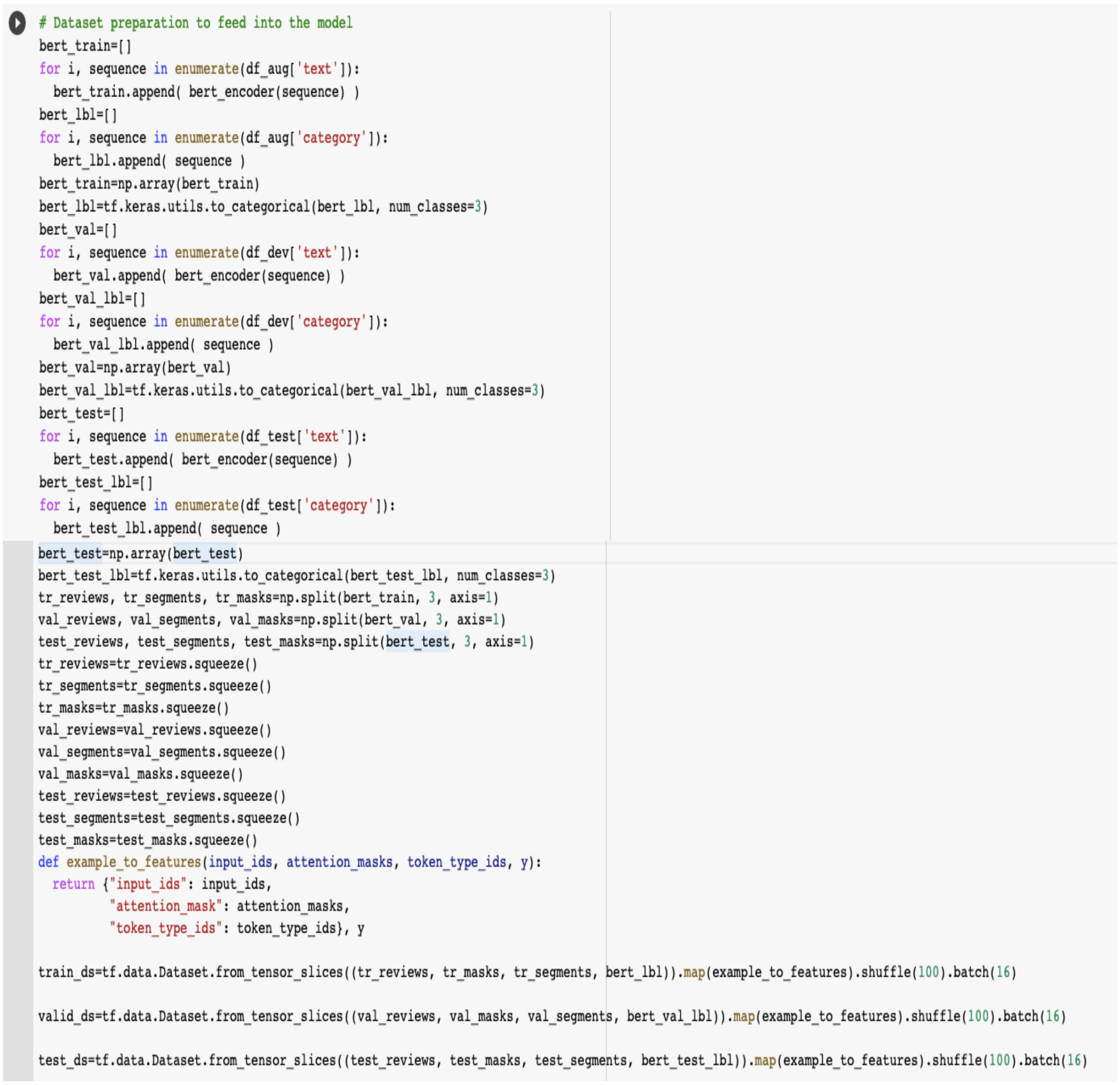
Figure 20 shows the code used to initialize RoBERTa Tokenizer



```
#Initialize a Roberta tokenizer to convert words tokens to numbers(vectorization)
bert_name= "roberta-base"
tokenizer=RobertaTokenizer.from_pretrained(bert_name,
                                           add_special_tokens=True,
                                           do_lower_case=False,
                                           max_length=180,
                                           pad_to_max_length=True)

def bert_encoder(review):
  encoded=tokenizer.encode_plus(review, add_special_tokens=True,
                                max_length=180, pad_to_max_length=True,
                                truncation=True,
                                return_attention_mask=True,
                                return_token_type_ids=True)
  return encoded['input_ids'], encoded['token_type_ids'], encoded['attention_mask']
```

Figure 20: RoBERTa Tokenization

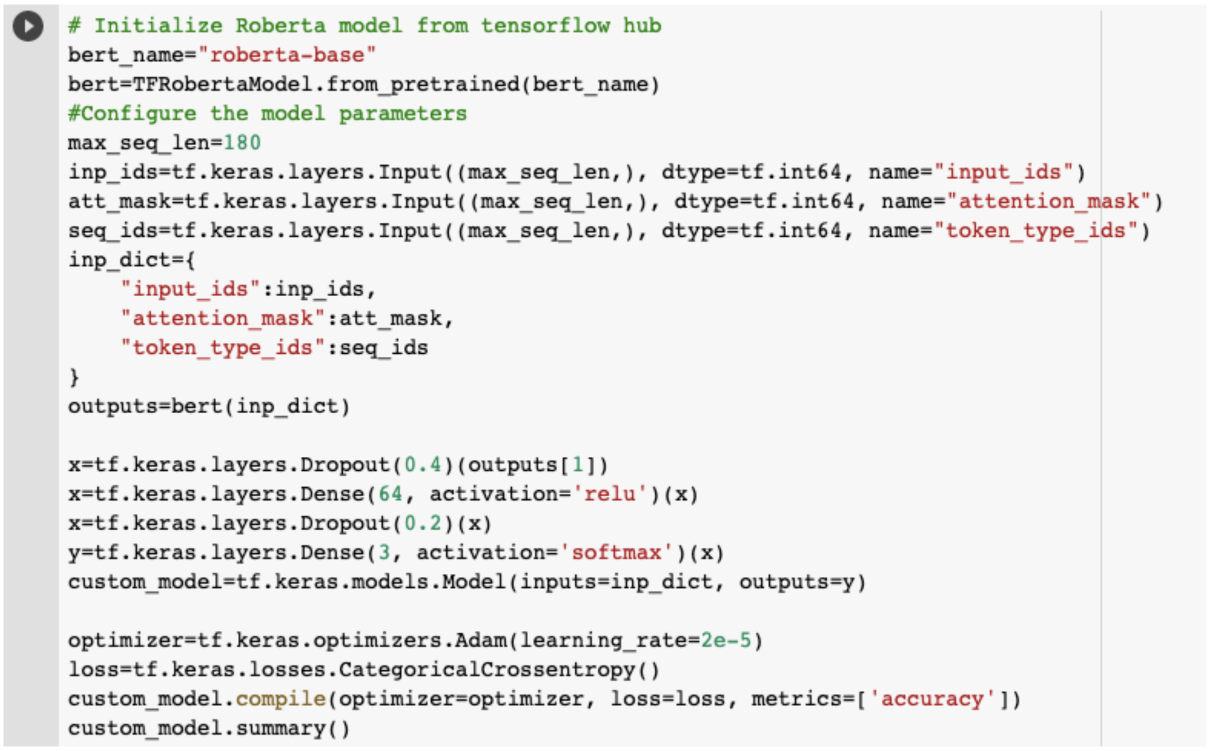
Figure 21 shows the code utilized to generate train, validation and test dataset in the RoBERTa model compatible format.



```
# Dataset preparation to feed into the model
bert_train=[]
for i, sequence in enumerate(df_aug['text']):
  bert_train.append( bert_encoder(sequence) )
bert_lbl=[]
for i, sequence in enumerate(df_aug['category']):
  bert_lbl.append( sequence )
bert_train=np.array(bert_train)
bert_lbl=tf.keras.utils.to_categorical(bert_lbl, num_classes=3)
bert_val=[]
for i, sequence in enumerate(df_dev['text']):
  bert_val.append( bert_encoder(sequence) )
bert_val_lbl=[]
for i, sequence in enumerate(df_dev['category']):
  bert_val_lbl.append( sequence )
bert_val=np.array(bert_val)
bert_val_lbl=tf.keras.utils.to_categorical(bert_val_lbl, num_classes=3)
bert_test=[]
for i, sequence in enumerate(df_test['text']):
  bert_test.append( bert_encoder(sequence) )
bert_test_lbl=[]
for i, sequence in enumerate(df_test['category']):
  bert_test_lbl.append( sequence )
bert_test=np.array(bert_test)
bert_test_lbl=tf.keras.utils.to_categorical(bert_test_lbl, num_classes=3)
tr_reviews, tr_segments, tr_masks=np.split(bert_train, 3, axis=1)
val_reviews, val_segments, val_masks=np.split(bert_val, 3, axis=1)
test_reviews, test_segments, test_masks=np.split(bert_test, 3, axis=1)
tr_reviews=tr_reviews.squeeze()
tr_segments=tr_segments.squeeze()
tr_masks=tr_masks.squeeze()
val_reviews=val_reviews.squeeze()
val_segments=val_segments.squeeze()
val_masks=val_masks.squeeze()
test_reviews=test_reviews.squeeze()
test_segments=test_segments.squeeze()
test_masks=test_masks.squeeze()
def example_to_features(input_ids, attention_masks, token_type_ids, y):
  return {"input_ids": input_ids,
          "attention_mask": attention_masks,
          "token_type_ids": token_type_ids}, y

train_ds=tf.data.Dataset.from_tensor_slices((tr_reviews, tr_masks, tr_segments, bert_lbl)).map(example_to_features).shuffle(100).batch(16)

valid_ds=tf.data.Dataset.from_tensor_slices((val_reviews, val_masks, val_segments, bert_val_lbl)).map(example_to_features).shuffle(100).batch(16)

test_ds=tf.data.Dataset.from_tensor_slices((test_reviews, test_masks, test_segments, bert_test_lbl)).map(example_to_features).shuffle(100).batch(16)
```

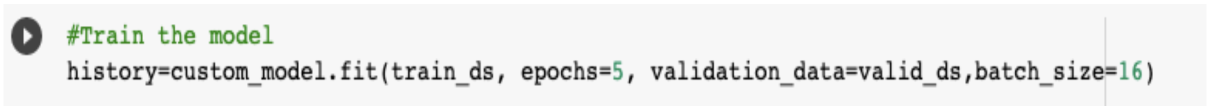
Figure 21: Generate train, valid and test dataset

Figure 22 shows the code utilized to initiate and configure the RoBERTa model. All the hyperparameters are defined in this step.



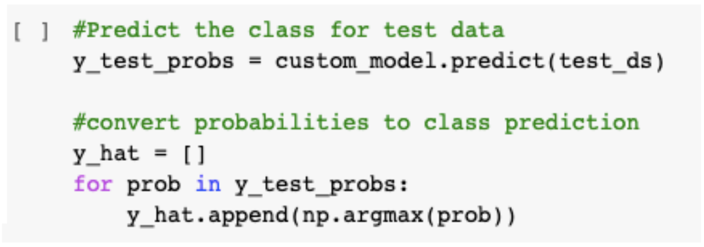
```
# Initialize Roberta model from tensorflow hub
bert_name="roberta-base"
bert=TFRobertaModel.from_pretrained(bert_name)
#Configure the model parameters
max_seq_len=180
inp_ids=tf.keras.layers.Input((max_seq_len,), dtype=tf.int64, name="input_ids")
att_mask=tf.keras.layers.Input((max_seq_len,), dtype=tf.int64, name="attention_mask")
seq_ids=tf.keras.layers.Input((max_seq_len,), dtype=tf.int64, name="token_type_ids")
inp_dict={
    "input_ids":inp_ids,
    "attention_mask":att_mask,
    "token_type_ids":seq_ids
}
outputs=bert(inp_dict)

x=tf.keras.layers.Dropout(0.4)(outputs[1])
x=tf.keras.layers.Dense(64, activation='relu')(x)
x=tf.keras.layers.Dropout(0.2)(x)
y=tf.keras.layers.Dense(3, activation='softmax')(x)
custom_model=tf.keras.models.Model(inputs=inp_dict, outputs=y)

optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5)
loss=tf.keras.losses.CategoricalCrossentropy()
custom_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
custom_model.summary()
```

Figure 22: Configure the RoBERTa model



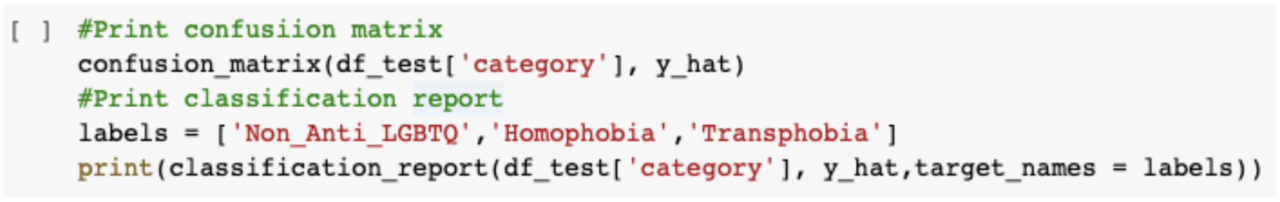
```
#Train the model
history=custom_model.fit(train_ds, epochs=5, validation_data=valid_ds,batch_size=16)
```

Figure 23: Model Training



```
[ ] #Predict the class for test data
    y_test_probs = custom_model.predict(test_ds)

    #convert probabilities to class prediction
    y_hat = []
    for prob in y_test_probs:
        y_hat.append(np.argmax(prob))
```

Figure 24: Predict the test data

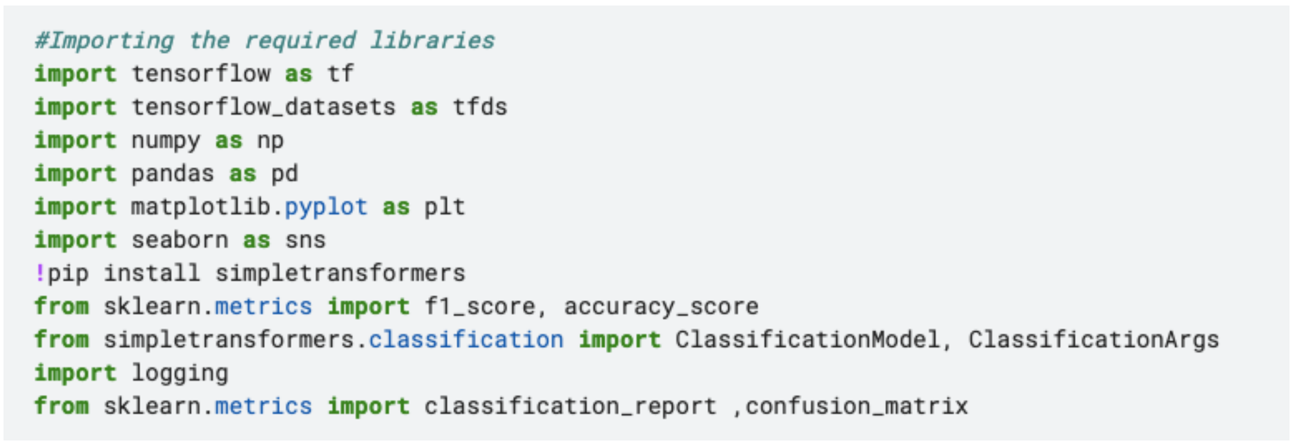


```
[ ] #Print confusiion matrix
    confusion_matrix(df_test['category'], y_hat)
    #Print classification report
    labels = ['Non_Anti_LGBTQ','Homophobia','Transphobia']
    print(classification_report(df_test['category'], y_hat,target_names = labels))
```

Figure 25: Display Confusion Matrix and Classification Report

4.3 mBERT

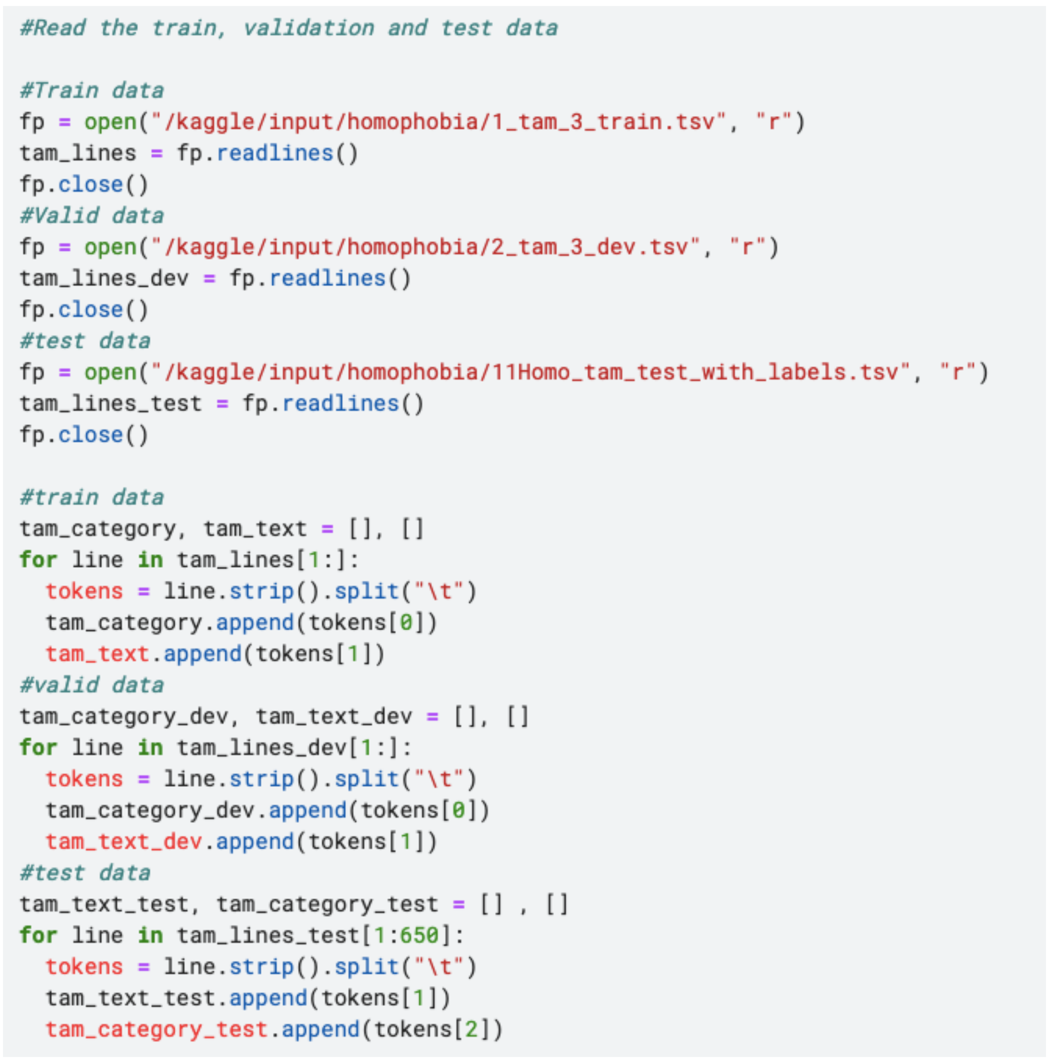
To implement the mBERT model on Tamil dataset, Kaggle Notebook was used to run the code. First the dataset is uploaded to Kaggle.



```python
#Importing the required libraries
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
!pip install simpletransformers
from sklearn.metrics import f1_score, accuracy_score
from simpletransformers.classification import ClassificationModel, ClassificationArgs
import logging
from sklearn.metrics import classification_report ,confusion_matrix
```

Figure 26: Importing all the required libraries

Figure 27 shows the code utilized to open the files and fetch labels and text data.



```python
#Read the train, validation and test data

#Train data
fp = open("/kaggle/input/homophobia/1_tam_3_train.tsv", "r")
tam_lines = fp.readlines()
fp.close()
#Valid data
fp = open("/kaggle/input/homophobia/2_tam_3_dev.tsv", "r")
tam_lines_dev = fp.readlines()
fp.close()
#test data
fp = open("/kaggle/input/homophobia/11Homo_tam_test_with_labels.tsv", "r")
tam_lines_test = fp.readlines()
fp.close()

#train data
tam_category, tam_text = [], []
for line in tam_lines[1:]:
  tokens = line.strip().split("\t")
  tam_category.append(tokens[0])
  tam_text.append(tokens[1])
#valid data
tam_category_dev, tam_text_dev = [], []
for line in tam_lines_dev[1:]:
  tokens = line.strip().split("\t")
  tam_category_dev.append(tokens[0])
  tam_text_dev.append(tokens[1])
#test data
tam_text_test, tam_category_test = [] , []
for line in tam_lines_test[1:650]:
  tokens = line.strip().split("\t")
  tam_text_test.append(tokens[1])
  tam_category_test.append(tokens[2])
```

Figure 27: Reading the data files

```
#Create pandas dataframe to store train, validation and test data
tam_data = { 'category': tam_category,
        'text': tam_text}
tam_df = pd.DataFrame(tam_data)


tam_data_dev = { 'category': tam_category_dev,
        'text': tam_text_dev}
tam_df_dev = pd.DataFrame(tam_data_dev)


tam_data_test = { 'category': tam_category_test,
                 'text': tam_text_test}
tam_df_test = pd.DataFrame(tam_data_test)
```

Figure 28: Store the train, valid and test data in pandas dataframe

```
#Label encoding the target variable
tam_df['category'] = tam_df['category'].replace('Non-anti-LGBT+ content', 0)
tam_df_dev['category'] = tam_df_dev['category'].replace('Non-anti-LGBT+ content', 0)
tam_df_test['category'] = tam_df_test['category'].replace('Non- anti LGBT content', 0)

tam_df['category'] = tam_df['category'].replace('Homophobic', 1)
tam_df_dev['category'] = tam_df_dev['category'].replace('Homophobic', 1)
tam_df_test['category'] = tam_df_test['category'].replace('homophobia', 1)

tam_df['category'] = tam_df['category'].replace('Transphobic', 2)
tam_df_dev['category'] = tam_df_dev['category'].replace('Transphobic', 2)
tam_df_test['category'] = tam_df_test['category'].replace('transphobia', 2)
```

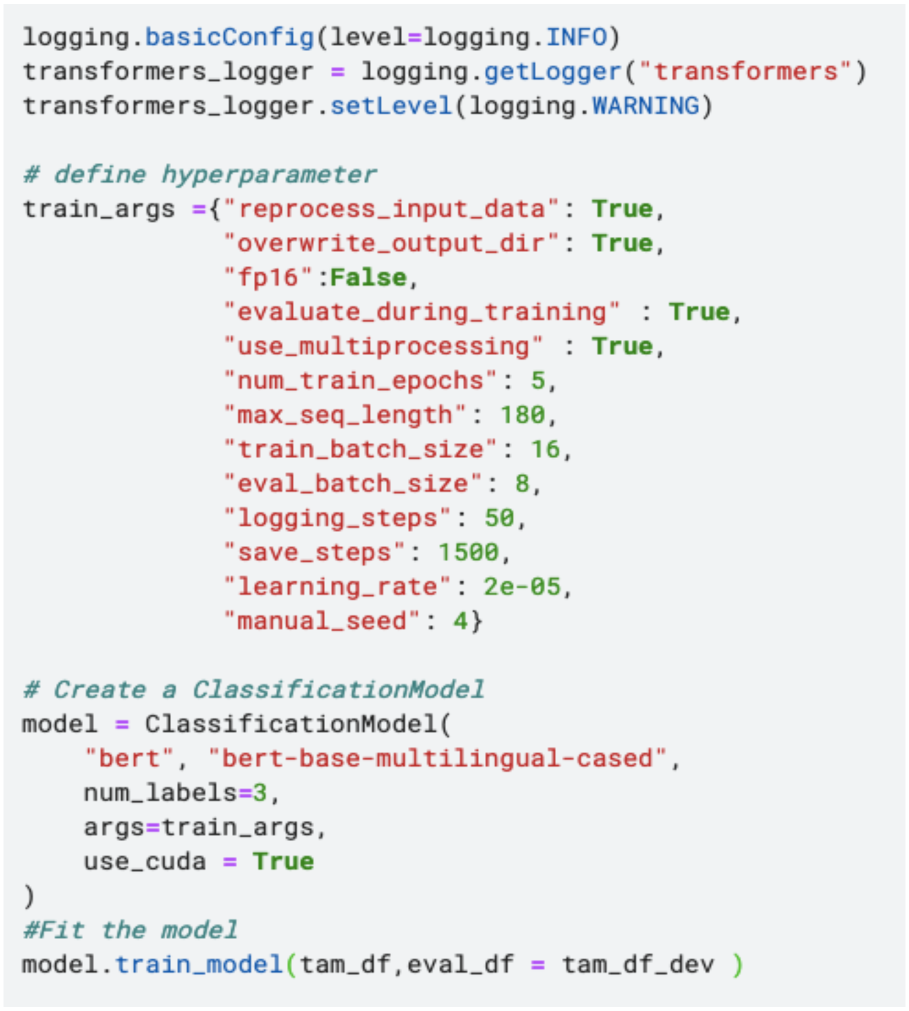
Figure 29: Perform label encoding on target variable

The dataset should be transformed into the format which is compatible with SimpleTransformers module. Columns were renamed as “text” and “labels”. Code used to execute this is shown in Figure 30.

```
#Transform the data according to the input format of simple transformers
tam_df=tam_df[["text", "category"]]
tam_df = tam_df.rename(columns={'text': 'text', 'category': 'labels'})
print(tam_df.head())

tam_df_dev=tam_df_dev[["text", "category"]]
tam_df_dev = tam_df_dev.rename(columns={'text': 'text', 'category': 'labels'})
print(tam_df_dev.head())

tam_df_test=tam_df_test[["text", "category"]]
tam_df_test = tam_df_test.rename(columns={'text': 'text', 'category': 'labels'})
print(tam_df_test.head())
```

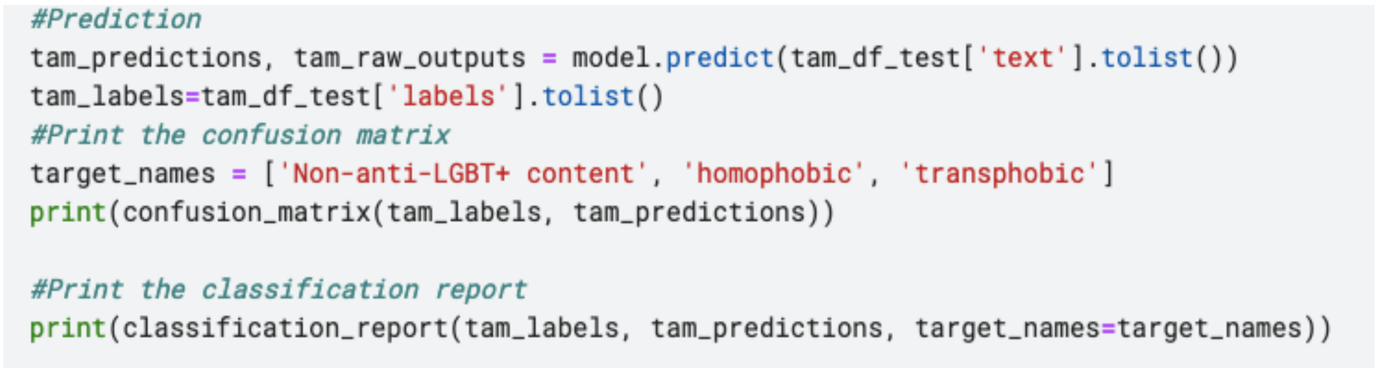
Figure 30: Transform the dataset to input format of the model

Figure 31 shows the code utilized to configure the model and train it on train dataset. Appropriate hyperparameters were defined in this step.



```
logging.basicConfig(level=logging.INFO)
transformers_logger = logging.getLogger("transformers")
transformers_logger.setLevel(logging.WARNING)

# define hyperparameter
train_args ={"reprocess_input_data": True,
             "overwrite_output_dir": True,
             "fp16":False,
             "evaluate_during_training" : True,
             "use_multiprocessing" : True,
             "num_train_epochs": 5,
             "max_seq_length": 180,
             "train_batch_size": 16,
             "eval_batch_size": 8,
             "logging_steps": 50,
             "save_steps": 1500,
             "learning_rate": 2e-05,
             "manual_seed": 4}

# Create a ClassificationModel
model = ClassificationModel(
    "bert", "bert-base-multilingual-cased",
    num_labels=3,
    args=train_args,
    use_cuda = True
)
#Fit the model
model.train_model(tam_df,eval_df = tam_df_dev )
```

Figure 31: Model building



```
#Prediction
tam_predictions, tam_raw_outputs = model.predict(tam_df_test['text'].tolist())
tam_labels=tam_df_test['labels'].tolist()
#Print the confusion matrix
target_names = ['Non-anti-LGBT+ content', 'homophobic', 'transphobic']
print(confusion_matrix(tam_labels, tam_predictions))

#Print the classification report
print(classification_report(tam_labels, tam_predictions, target_names=target_names))
```

Figure 32: Predict the test data

References

Chakravarthi, B.R., Priyadharshini, R., Ponnusamy, R., Kumar, P., Sampath, K., Thenmozhi, D., Thangasamy, S., McCrae, J.P., n.d. Dataset for Identification of Homophobia and Transophobia in Multilingual YouTube Comments. Nat. Lang. Eng. 44.

Wei, J., Zou, K., 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Presented at the Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, pp. 6381–6387. https://doi.org/10.18653/v1/D19-1670