TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
NGÀNH TOÁN-TIN HỌC

PYTHON CHO KHOA HỌC DỮ LIỆU

FINAL SEMESTER PROJECT
REALTIME SENTIMENT ANALYSYS DASHBOARD

Giảng viên phụ trách
Hà Văn Thảo

MỤC LỤC

Lưu ý báo cáo

Thưa thầy, vì do file code của nhóm em không chỉ có mỗi file ibnpy mà còn có code của file py trên vscode chúng em không thể chụp lại được vì nó quá dài. Nên thầy thông cảm giúp nhóm chúng em khi phải paste code trên doc ạ. Tụi em xin chịu mọi trách nhiệm nếu code trong doc khác với code trong file code của tụi em ạ. Em cảm on thầy

Thầy có thể xem file code tổng của tụi em ở link github dưới đây ạ

https://github.com/Amature123/new_new_sentiment

INTRODUCE

1.1 Thông tin nhóm

• Thành viên:
– 22110191 - Nguyễn Thành Tài
– 22110212 - Nguyễn Quang Thịnh
– 22110068 - Lê Kim Hùng

1.2 Nguyên tắc hoạt động nhóm

• Ý tưởng được đóng góp từ tất cả thành viên của nhóm.
• Mọi người tôn trọng ý kiến của nhau.
• Giao và làm việc đúng thời hạn đưa ra. Nếu trễ thì sẽ chịu trách nhiệm về phần của mình.

1.3 Mục tiêu project

Với mục tiêu làm một pipeline đầy đủ về các chức năng cũng như vai trò của data scientist, nhóm em đã thực hiện project Sentiment Analysys dashboard. Project tập trung về quá trình train model và cũng như cách để deploy model lấy tiếp dữ liệu từ trang web voz.vn để có thể visualize và analyze data.

1.4 Data source

- Vietnamese sentiment dataset: Chứa dữ liệu của đánh giá khách hàng, số sao đánh giá, và tình trạng của câu nói https://www.kaggle.com/datasets/linhlpv/vietnamese-sentiment-analyst
- Voz.vn: Nơi scrap các data về để chạy thử mô hình https://voz.vn/whats-new/

Architecture

2.1 Dictionary tree

Ở phần này chúng em chỉ liệt kê các thành phần chính trong code. Các phần khác của code đã tự động tạo khi khai triển 1 dự án của fullstack .

- Vite_project: Nơi chứa các code để làm frontend
  - "node_modules": File dependencies của Nodejs
  - “src”: Code chính của frontend
    - App.jsx: File chính tạo backend
  - “Dockerfile”: tạo container cho frontend

- Voz_neww: Backend của project
  - “model”: file model để deploy dự án
  - “postgres”: file tạo database schema mẫu
  - “Voz_neww”: Khung sườn của backend
    - pipeline.py: Ống sự kiện, nơi quản lý luốn dữ liệu
    - spiders/demospider.py: Code scraping data
  - “Dockerfile”:tạo container cho backend
  - “main.py”: file code khởi tạo backend
  - “output.json”: file data mẫu do Voz_neww scrap về
  - “requirements.txt”: file các thư viện cần thiết cho python

- Docker-compose.yaml: Xây dựng các container cho docker và chạy code tự động

- Các version pipeline: sự thay đổi của code trong Voz_neww cho các phiên bản của model khác nhau

- test.py: Mock code

bithon

January 5, 2025

[29]:
```
# Import các thư viện cần thiết
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import re
# Đảm bảo in hết tất cả các câu nói
pd.set_option('display.max_rows', None)  # Hiển thị tất cả các dòng
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).
```

[30]:
```
df = pd.read_csv('/content/drive/MyDrive/test_py/data_sent.csv')
df.head()
```

[30]:

	comment	label	rate	Unnamed: 3
0	Áo bao đẹp ạ!!	POS	5	NaN
1	Tuyệt vời !	POS	5	NaN
2	2day ao khong giong trong.	NEG	1	NaN
3	Mùi thơm,bôi lên da mềm da.	POS	5	NaN
4	Vải đẹp, dày dặn.	POS	5	NaN

[31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31460 entries, 0 to 31459
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   comment     31460 non-null  object
 1   label       31460 non-null  object
 2   rate        31460 non-null  int64
 3   Unnamed: 3  23 non-null     object
dtypes: int64(1), object(3)
memory usage: 983.2+ KB
```

[32]:
```
df.drop(['rate','Unnamed: 3'],axis=1,inplace=True)
```

[33]:
```
label_mapping = {
    'POS': "tiêu cực",
    'NEU': "trung lập",
    'NEG': "tích cực"
}
df['label'] = df['label'].map(label_mapping)#tranform label from -1, 0, 1 to␣
 ↪"tiêu cực", "trung lập" and "tích cực"
```

Đoạn code này thực hiện một số bước tiền xử lý dữ liệu và gán nhãn cảm xúc cho tập dữ liệu, cụ thể như sau:

- Tóm tắt
- Đoạn mã này thực hiện tiền xử lý dữ liệu, bao gồm việc đọc dữ liệu từ file CSV và hiển thị thông tin về dữ liệu.

[35]:
```
df.isna().sum()
```

[35]:
```
comment    0
label      0
dtype: int64
```

[34]:
```
#Check again
df.head()
```

[34]:

	comment	label
0	Áo bao đẹp ạ!!	1
1	Tuyệt vời !	1
2	2day ao khong giong trong.	-1
3	Mùi thơm,bôi lên da mềm da.	1
4	Vải đẹp, dày dặn.	1

[36]:
```
def clean_comments(text):
  text = re.sub(r'[!?]', '', text)
  text = re.sub(r'[^\w\s]', '', text)
  return text

df['comment'] = df['comment'].apply(clean_comments)
```

[37]:
```
df.head(10)
```

[37]:

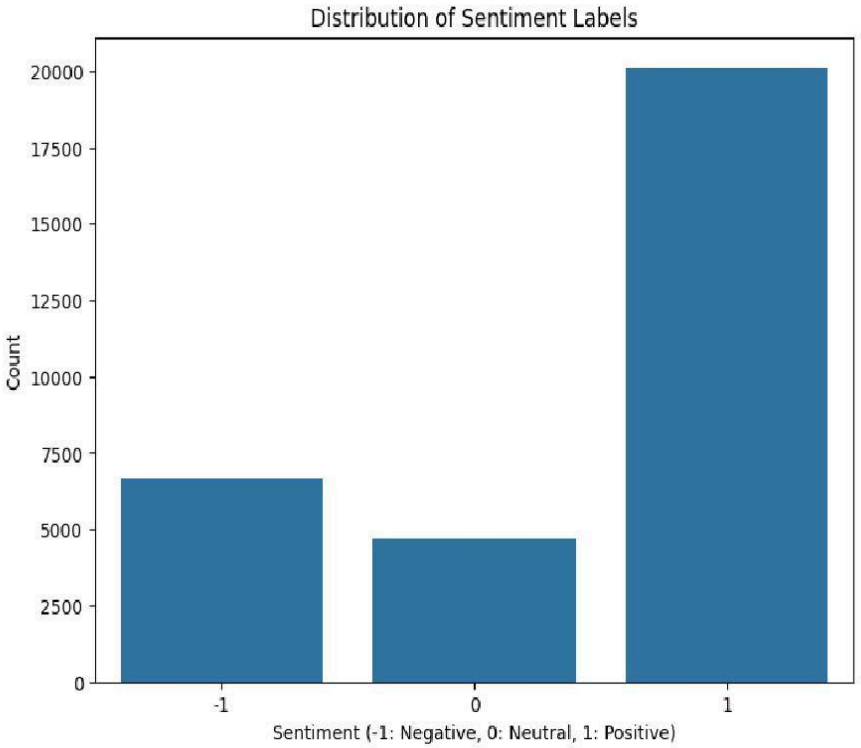
	comment	label
0	Áo bao đẹp ạ	1
1	Tuyệt vời	1
2	2day ao khong giong trong	-1
3	Mùi thơmbôi lên da mềm da	1
4	Vải đẹp dày dặn	1

```
5      Hàng rất đẹp rất chi là ưng ý      1
6   Chất lượng sản phẩm tốt date dài      1
7      Ăn nói và thái độ phục vụ tốt      1
8        Đóng gói sản phẩm chắc chắn      1
9  tất sờn hết ca chưa dùng mà vay r     -1
```

1 EDA là gì

- EDA (Exploratory Data Analysis) là quá trình phân tích dữ liệu khám phá, được sử dụng để hiểu rõ hơn về cấu trúc, mối quan hệ và đặc điểm của dữ liệu trước khi áp dụng các mô hình thống kê hoặc học máy.
- Mục tiêu chính của EDA:
- Hiểu rõ dữ liệu: Phát hiện phân phối, xu hướng, và các mẫu trong dữ liệu.
- Phát hiện bất thường: Xác định dữ liệu thiếu, ngoại lệ (outliers).
- Xác định mối quan hệ: Tìm mối tương quan giữa các biến.
- Kiểm tra giả định: Đảm bảo dữ liệu phù hợp cho mô hình hóa.
- Các bước thực hiện EDA:
- Tổng quan dữ liệu: Kiểm tra kích thước, kiểu dữ liệu, giá trị trùng lặp.
- Thống kê mô tả: Tính toán giá trị trung bình, trung vị, phương sai, độ lệch chuẩn.
- Trực quan hóa dữ liệu: Biểu đồ histogram, boxplot, scatter plot, heatmap.
- Phân tích tương quan: Ma trận tương quan giữa các biến số.
- Xử lý dữ liệu thiếu hoặc ngoại lệ: Đưa ra chiến lược xử lý.
- Công cụ phổ biến:
- Python: Pandas, Matplotlib, Seaborn.
- R: ggplot2, dplyr.

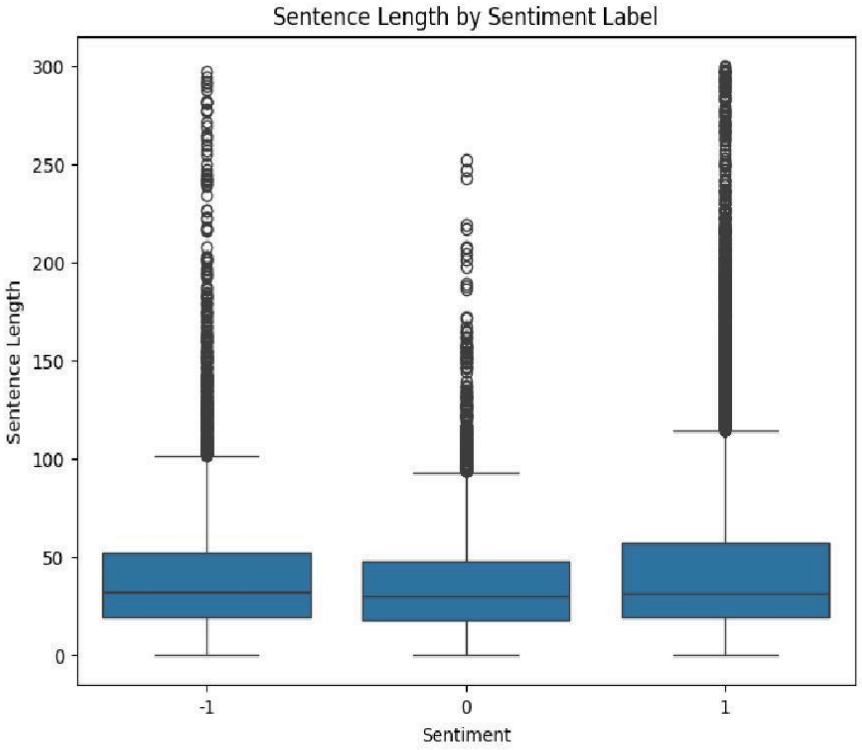
[39]:
```
plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=df)
plt.title("Distribution of Sentiment Labels")
plt.xlabel("Sentiment (-1: Negative, 0: Neutral, 1: Positive)")
plt.ylabel("Count")
plt.show()
```



Đoạn code dưới đây sử dụng thư viện matplotlib và seaborn để tạo biểu đồ đếm (count plot) hiển thị phân phối của các nhãn cảm xúc trong dữ liệu. how(): Hiển thị biểu đồ lên màn hình.

- Tóm tắt
- Đoạn code vẽ một biểu đồ đếm để hiển thị phân phối các nhãn cảm xúc trong cột label của DataFrame. Biểu đồ này giúp người dùng thấy rõ số lượng câu thuộc vào các nhóm cảm xúc Tiêu cực, Trung tính và Tích cực.

[40]:
```
# 5. Thêm cột độ dài câu nói
df['text_length'] = df['comment'].apply(len)

# Vẽ biểu đồ hộp (boxplot) để xem phân phối độ dài câu nói theo nhãn cảm xúc
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='text_length', data=df)
plt.title("Sentence Length by Sentiment Label")
plt.xlabel("Sentiment")
plt.ylabel("Sentence Length")
plt.show()
```



Đoạn code dưới đây thực hiện việc thêm một cột mới để tính độ dài của các câu nói trong dữ liệu, sau đó vẽ một biểu đồ hộp (boxplot) để phân tích sự phân phối độ dài câu nói theo từng nhãn cảm xúc.

- Tóm tắt
- Đoạn mã này thêm một cột mới vào DataFrame để tính độ dài của các câu trong cột message. Sau đó, nó vẽ một biểu đồ hộp (boxplot) để phân tích sự phân phối độ dài của các câu nói theo nhãn cảm xúc. Biểu đồ này giúp ta hiểu rõ hơn về sự khác biệt trong độ dài câu giữa các nhóm cảm xúc khác nhau (tiêu cực, trung tính, tích cực).

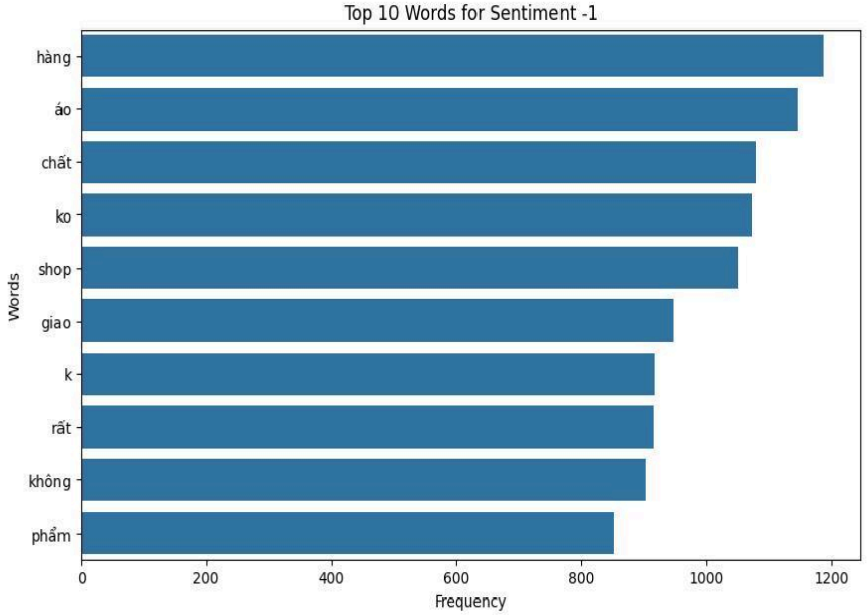
[41]:
```
# 6. Đếm từ phổ biến cho từng nhãn cảm xúc và vẽ biểu đồ
def plot_top_words(data, sentiment_value, n_top=10):
    # Lọc câu nói theo nhãn cảm xúc
    text_data = " ".join(data[data['label'] == sentiment_value]['comment'])

    # Loại bỏ các ký tự không phải chữ cái và chuyển thành chữ thường
    words = re.findall(r'\b\w+\b', text_data.lower())

    # Đếm tần suất xuất hiện của các từ
    word_counts = Counter(words)
```

```
    # Lấy n từ phổ biến nhất
    common_words = word_counts.most_common(n_top)
    words, counts = zip(*common_words)

    # Vẽ biểu đồ cột (bar plot) cho các từ phổ biến nhất
    plt.figure(figsize=(10, 6))
    sns.barplot(x=counts, y=words)
    plt.title(f"Top {n_top} Words for Sentiment {sentiment_value}")
    plt.xlabel("Frequency")
    plt.ylabel("Words")
```

Đoạn code dưới đây thực hiện việc đếm và trực quan hóa các từ phổ biến trong các câu nói theo từng nhãn cảm xúc, với mục đích giúp người dùng hiểu rõ hơn về từ vựng đặc trưng của các nhóm cảm xúc.

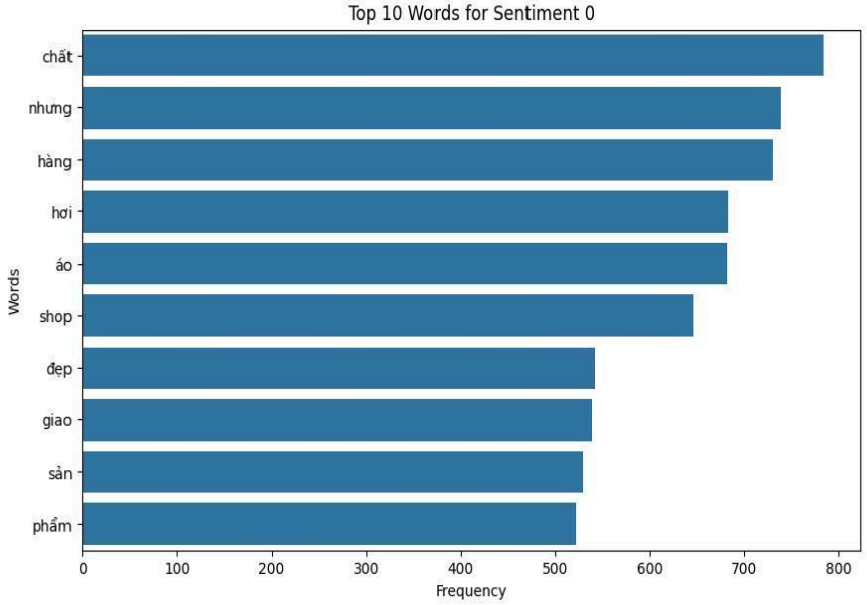
- Tóm tắt
- Hàm plot_top_words() nhận vào dữ liệu và một nhãn cảm xúc cụ thể, sau đó lọc các câu nói theo nhãn cảm xúc đó.
- Nó xử lý văn bản để tìm ra các từ phổ biến, đếm tần suất của chúng và vẽ biểu đồ cột để trực quan hóa các từ phổ biến nhất.
- Biểu đồ này giúp người dùng hiểu rõ hơn về những từ hay xuất hiện trong các câu nói thuộc mỗi nhóm cảm xúc (tiêu cực, trung tính, tích cực).

[42]:
```
plot_top_words(df, -1)  # Negative
```

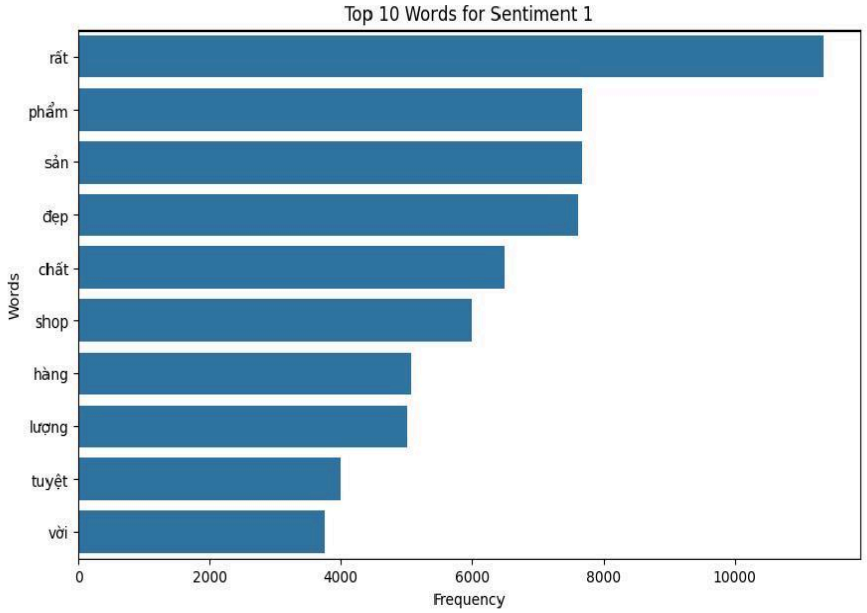


- Tóm lại:
- Câu lệnh này gọi hàm plot_top_words để vẽ biểu đồ thể hiện các từ phổ biến nhất trong các câu có nhãn cảm xúc tiêu cực (-1) từ DataFrame df.

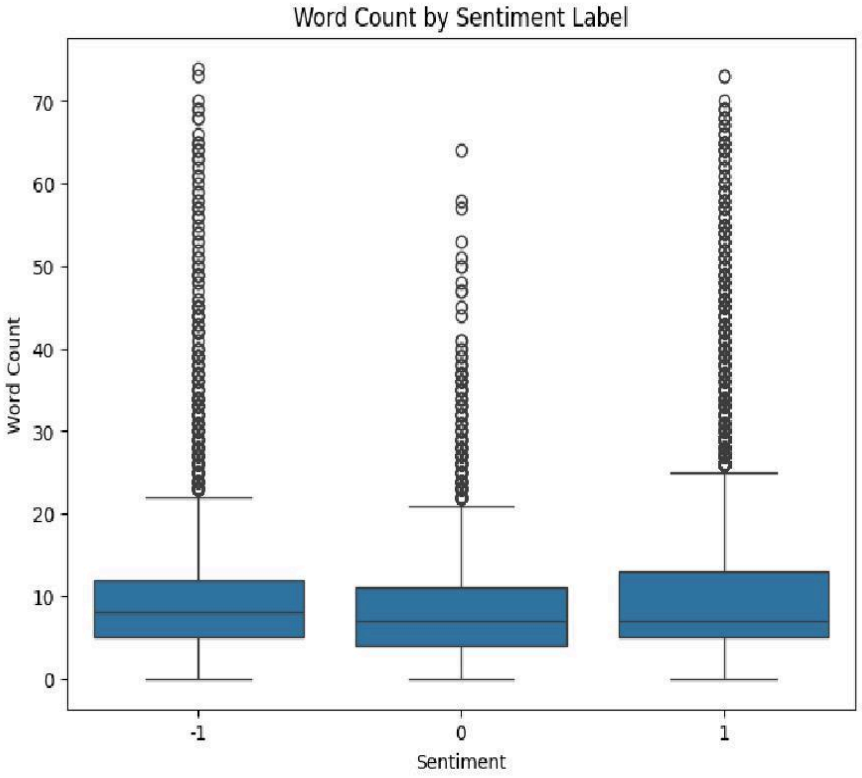
[43]:
```
plot_top_words(df, 0)    # Neutral
```



[44]:
```
plot_top_words(df, 1)    # Positive
```



[45]:
```python
# 7. Thêm cột đếm số từ trong mỗi câu
df['word_count'] = df['comment'].apply(lambda x: len(x.split()))

# Vẽ biểu đồ hộp (boxplot) để xem phân phối số từ theo nhãn cảm xúc
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='word_count', data=df)
plt.title("Word Count by Sentiment Label")
plt.xlabel("Sentiment")
plt.ylabel("Word Count")
plt.show()

# Tính số lượng từng nhãn cảm xúc
label_counts = df['label'].value_counts()
```

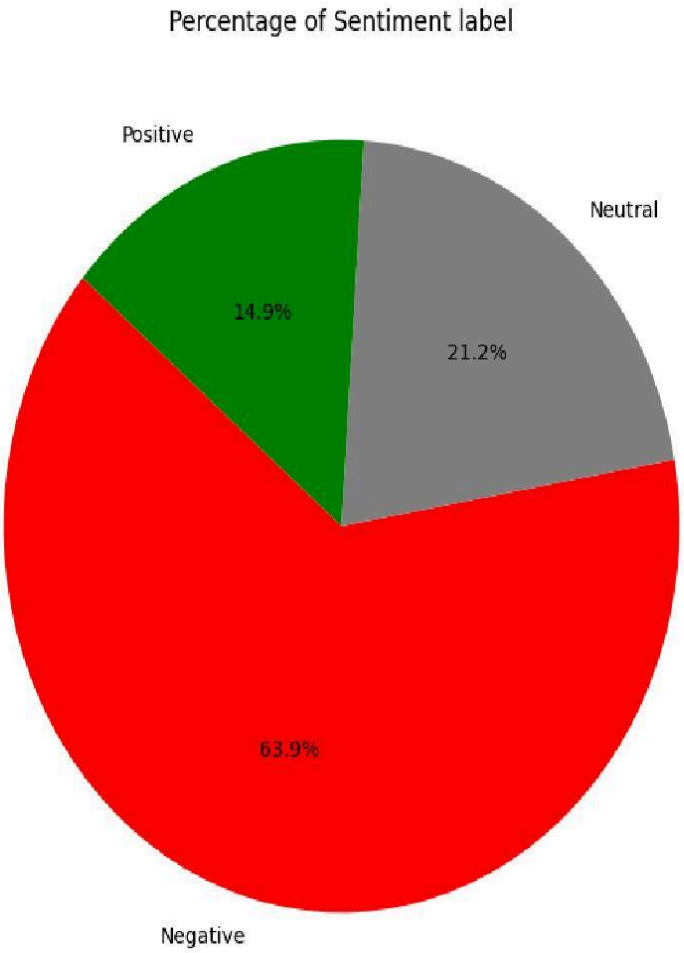


Đoạn code dưới đây thực hiện việc tính số lượng từ trong mỗi câu, sau đó vẽ biểu đồ hộp (boxplot) để phân tích sự phân phối số từ theo các nhãn cảm xúc. Cuối cùng, nó tính số lượng các câu tương ứng với từng nhãn cảm xúc trong dữ liệu.

- Tóm tắt:
- Đoạn code này tính số lượng từ trong mỗi câu và thêm thông tin này vào một cột mới.
- Sau đó, nó vẽ biểu đồ hộp (boxplot) để phân tích sự phân phối số từ theo từng nhãn cảm xúc. Cuối cùng, nó tính số lượng câu cho mỗi nhãn cảm xúc, giúp bạn hiểu được phân bố cảm xúc trong tập dữ liệu.

[46]:
```
# Vẽ biểu đồ tròn
plt.figure(figsize=(8, 8))
plt.pie(
    label_counts,
    labels=['Negative', 'Neutral', 'Positive'],  # Nhãn tương ứng
    autopct='%1.1f%%',  # Hiển thị phần trăm
    startangle=140,  # Bắt đầu từ góc 140 độ
    colors=['red', 'grey', 'green']  # Màu sắc cho từng nhãn
)
plt.title("Percentage of Sentiment label")
```

```
plt.show()
```



Đoạn code dưới đây sử dụng thư viện matplotlib để vẽ biểu đồ tròn (pie chart), hiển thị tỷ lệ phần trăm của các nhãn cảm xúc trong dữ liệu.

- Tóm tắt
- Đoạn mã này sẽ vẽ một biểu đồ tròn để hiển thị tỷ lệ phần trăm của các nhãn cảm xúc Tiêu cực, Trung tính và Tích cực trong dữ liệu.
- Biểu đồ tròn sẽ có màu sắc đặc biệt cho từng nhãn cảm xúc, giúp người xem dễ dàng nhận diện phân phối cảm xúc trong tập dữ liệu.

2 First model for the project

3 Load lib

[ ]:
```
!pip install pyvi > /dev/null 2>&1 #install without notification
import numpy as np
import pandas as pd
import tensorflow as tf
import pickle
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Embedding, Dense, Dropout, Bidirectional,␣
↪LSTM, GRU, Input,MaxPooling1D, GlobalMaxPooling1D, LayerNormalization, Conv1D
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from pyvi import ViTokenizer
from pyvi import ViUtils
```

4 Load data

4.1 data set already saved as the link below:

https://raw.githubusercontent.com/hungitnoi/model-for-project/refs/heads/master/s.csv

5 Data preparation and preprocessing for tranning

5.1 data separation as Input_label and input_data

5.1.1 Input_label: target variable

[ ]:
```
Input_label = df['label']
Input_label
```

[ ]:
```
0        tích cực
1        tích cực
2        tiêu cực
3        tích cực
4        tích cực
          ...
31455    tiêu cực
31456    tích cực
31457    tích cực
31458    tích cực
```

```
31459    tích cực
Name: label, Length: 31460, dtype: object
```

5.1.2 input_data: data features

[ ]:
```
input_data = df['comment']
input_data
```

[ ]:
```
0                                          Áo bao đẹp ạ!!
1                                             Tuyệt vời !
2                                2day ao khong giong trong.
3                               Mùi thơm,bôi lên da mềm da.
4                                         Vải đẹp, dày dặn.
                                ...
31455                                     Không đáng tiền.
31456                                        Quần rất đẹp.
31457                                Hàng đẹp đúng giá tiền.
31458                                     Chất vải khá ổn.
31459    áo rất ok nhé , vải mịn , len cao cổ này phối ...
Name: comment, Length: 31460, dtype: object
```

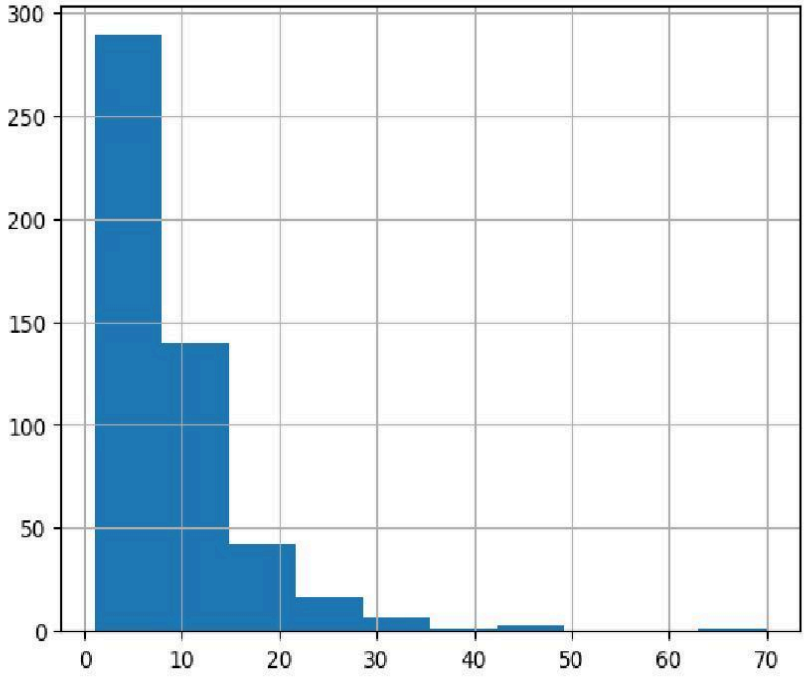
###Processing data

[ ]:
```
#In here, we use some basic func to preprocessing data for the model
#Cause Vietnamese has accents so we use ViTokenizer in libraly pyvi
label_dict = {'tiêu cực':0, 'trung lập':1, 'tích cực':2}

input_pre=[]
label_with_accent=[]
for idx,dt in enumerate(input_data):
   input_text_pre=list(tf.keras.preprocessing.text.text_to_word_sequence(dt))
   input_text_pre=" ".join(input_text_pre)
   input_text_pre_accent=ViTokenizer.tokenize(input_text_pre)
   input_pre.append(input_text_pre_accent)
   label_with_accent.append(Input_label[idx])

#After processing, the data has basically been processed at the most basic␣
 ↪level for the model.
```

###After processing data, we start visualize length of sentencs for next steps

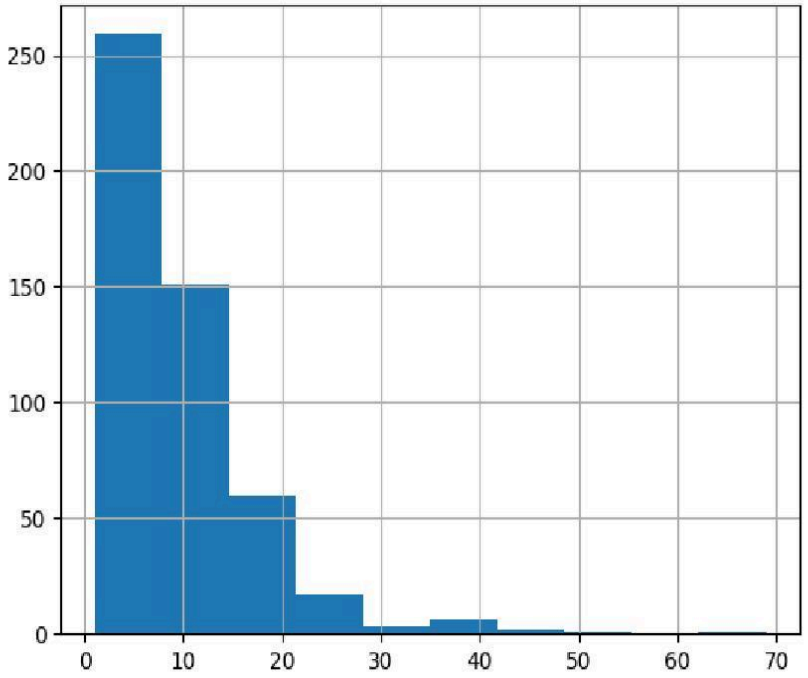
[ ]:
```
seq_len= [len(i.split()) for i in input_pre[0:500]]
pd.Series(seq_len).hist(bins=10)
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



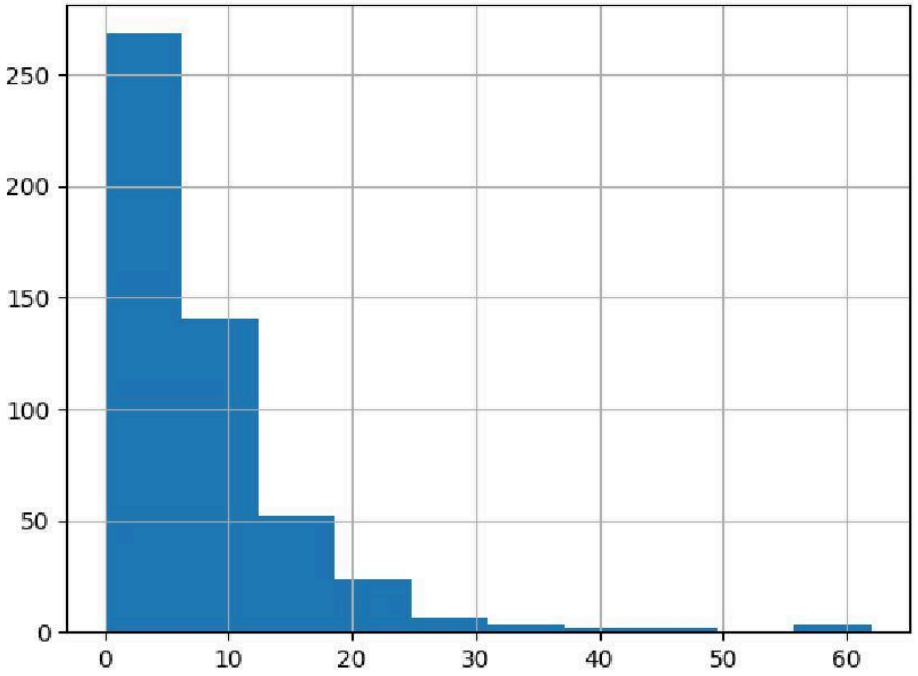
[ ]:
```
seq_len= [len(i.split()) for i in input_pre[500:1000]]
pd.Series(seq_len).hist(bins=10)
plt.show
```

[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



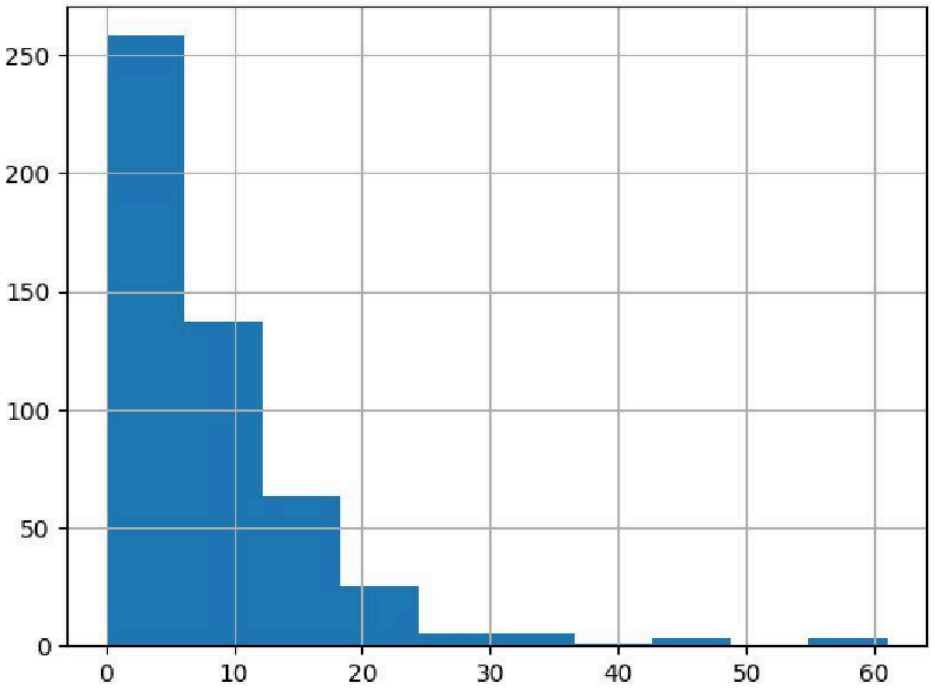

```
[ ]: seq_len= [len(i.split()) for i in input_pre[1000:1500]]
     pd.Series(seq_len).hist(bins=10)
     plt.show
```

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```



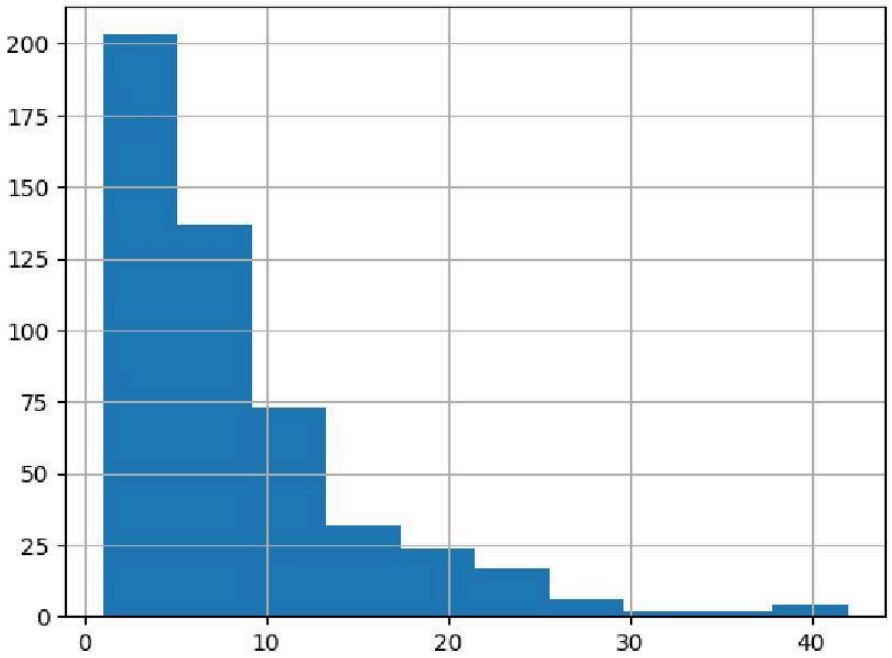
[ ]:
```
seq_len= [len(i.split()) for i in input_pre[1500:2000]]
pd.Series(seq_len).hist(bins=10)
plt.show
```

[ ]: <function matplotlib.pyplot.show(close=None, block=None)>

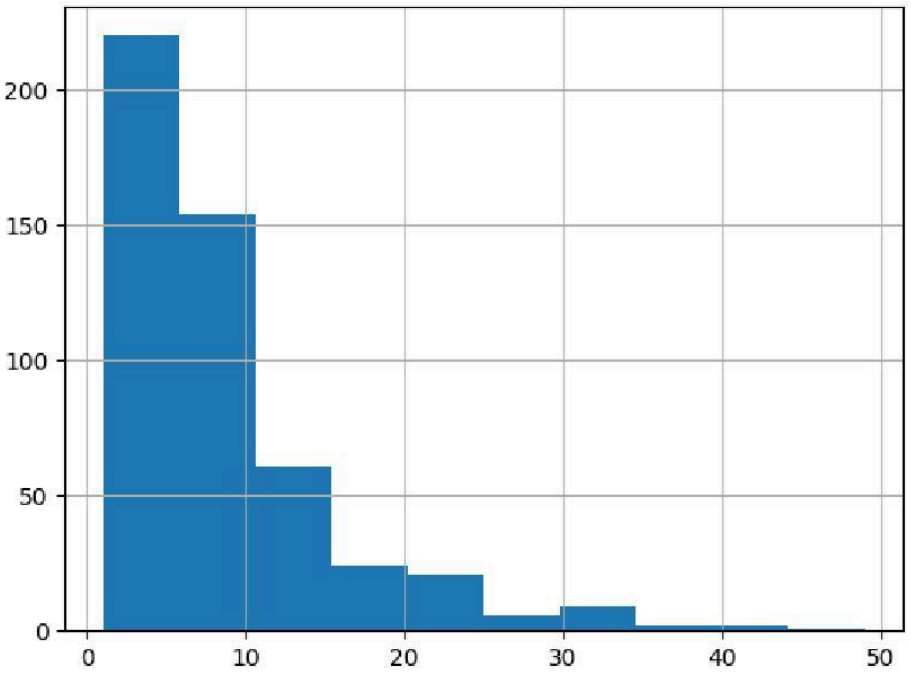


[ ]:
```
seq_len= [len(i.split()) for i in input_pre[2000:2500]]
pd.Series(seq_len).hist(bins=10)
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[ ]:
```
seq_len= [len(i.split()) for i in input_pre[2500:3000]]
pd.Series(seq_len).hist(bins=10)
plt.show()
```



[ ]:
```
#As we can see in 6 charts, an average sentence has about 335 words, so we use␣
 ↪it!
lable_idx=[label_dict[i] for i in label_with_accent]
lable_tf=tf.keras.utils.to_categorical(lable_idx,num_classes=3)

#In here, we use Tokenizer of tensorflow, running by input_pre that already␣
 ↪processed with ViTokenizer
tokenizer_data=Tokenizer(oov_token='<OOV>', filters='',split= ' ')
tokenizer_data.fit_on_texts(input_pre)

tokenized_data_text = tokenizer_data.texts_to_sequences(input_pre)
#Now, every single word in the sentence will map to a vector of numbers with␣
 ↪maxlen is 335
vec_data = pad_sequences(tokenized_data_text, padding='post', maxlen=335)


#Use pickle.dump to save it to reuse in fulture
pickle.dump(tokenizer_data, open("tokenizer_data.pkl", "wb"))

print("input data.shape",vec_data.shape)
data_vocab_size=len(tokenizer_data.word_index)+1
print("data_vocab_size",data_vocab_size)
```

```
#Use 80% for Tranning, 20% for Validation
x_train,x_val,y_train,y_val=train_test_split(vec_data,lable_tf,test_size=0.
↪2,random_state=42)
#Continue to cut out 10% for Test sample, and the rest will be our real training
x_train,x_test,y_train,y_test=train_test_split(x_train,y_train,test_size=0.
↪1,random_state=42)


print("Training sample",len(x_train))
print("Validation sample",len(x_val))
print("Test sample",len(x_test))
```

```
input data.shape (31460, 335)
data_vocab_size 7814
Training sample 22651
Validation sample 6292
Test sample 2517
```

##Create Model with CNN(Convolutional Neural Network) and Bidirectional

[ ]:
```
def generate_model():
    #Sets the dropout rate to prevent overfitting by randomly deactivating 40%␣
↪of neurons during training.
    dropout_threshold = 0.4

    #Vocabulary size for the embedding layer. This depends on the dataset.
    input_dim = data_vocab_size

    output_dim = 32 #The dimensionality of the embedding vectors.
    input_length = 335 #equivalent to maxlen in the padding process creating␣
↪vector sets

    #Initializes weights using a variance-scaling method (good for deep␣
↪learning).
    initializer = tf.keras.initializers.GlorotNormal()

    input_layer = Input(shape=(input_length,))#Defines the input shape as␣
↪(335,), a sequence of tokens with a fixed length of 335.

    #Embedding: Maps input tokens (integers) into dense vectors of size 32.
    feature = Embedding(input_dim=input_dim, output_dim=output_dim,
                        input_length=input_length,␣
↪embeddings_initializer="GlorotNormal")(input_layer)

    #After having Embedding feature, we split it into 2 branches CNN and␣
↪Bidirectional
```

```
  #Use CNN branch to extract the information
  cnn_feature = Conv1D(filters=32, kernel_size=3, padding='same',␣
↪activation='relu')(feature)
  cnn_feature = MaxPooling1D()(cnn_feature)
  cnn_feature = Dropout(dropout_threshold)(cnn_feature)
  cnn_feature = Conv1D(filters=32, kernel_size=3, padding='same',␣
↪activation='relu')(cnn_feature)
  cnn_feature = MaxPooling1D()(cnn_feature)
  cnn_feature = LayerNormalization()(cnn_feature)
  cnn_feature = Dropout(dropout_threshold)(cnn_feature)

  #Use Bidirectional
  bi_lstm_feature = Bidirectional(LSTM(units=32, dropout=dropout_threshold,␣
↪return_sequences=True, kernel_initializer=initializer),␣
↪merge_mode='concat')(feature)
  bi_lstm_feature = MaxPooling1D()(bi_lstm_feature)

  bi_lstm_feature = Bidirectional(GRU(units=32, dropout=dropout_threshold,␣
↪return_sequences=True, kernel_initializer=initializer),␣
↪merge_mode='concat')(bi_lstm_feature)
  bi_lstm_feature = MaxPooling1D()(bi_lstm_feature)
  bi_lstm_feature = LayerNormalization()(bi_lstm_feature)


  #Use Concatenate to synthesize cnn_feature and bi_lstm_feature to only one␣
↪layer
  combine_feature = tf.keras.layers.Concatenate()([cnn_feature,␣
↪bi_lstm_feature])
  combine_feature = GlobalMaxPooling1D()(combine_feature)#Use␣
↪GlobalMaxPooling1D to synthesize into 1 fully connected
  combine_feature = LayerNormalization()(combine_feature)

  #Go through classifiers to proceed with classification
  classifier = Dense(90, activation='relu')(combine_feature)
  classifier = Dropout(0.2)(classifier)
  classifier = Dense(70, activation='relu')(classifier)
  classifier = Dropout(0.2)(classifier)
  classifier = Dense(50, activation='relu')(classifier)
  classifier = Dropout(0.2)(classifier)
  classifier = Dense(30, activation='relu')(classifier)
  classifier = Dropout(0.2)(classifier)
  #problem for 3 class classifier so the final output is 3, activation is␣
↪softmax
  classifier = Dense(3, activation='softmax')(classifier)
  model = tf.keras.Model(inputs=input_layer, outputs=classifier)
```

```
    return model

model = generate_model()
adam = Adam(learning_rate=0.001)
model.compile(optimizer=adam, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 335)	0	-
embedding (Embedding)	(None, 335, 32)	250,048	input_layer[0][0]
conv1d (Conv1D)	(None, 335, 32)	3,104	embedding[0][0]
max_pooling1d (MaxPooling1D)	(None, 167, 32)	0	conv1d[0][0]
dropout (Dropout)	(None, 167, 32)	0	max_pooling1d[0][0]
bidirectional (Bidirectional)	(None, 335, 64)	16,640	embedding[0][0]
conv1d_1 (Conv1D)	(None, 167, 32)	3,104	dropout[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 167, 64)	0	bidirectional[0][0]

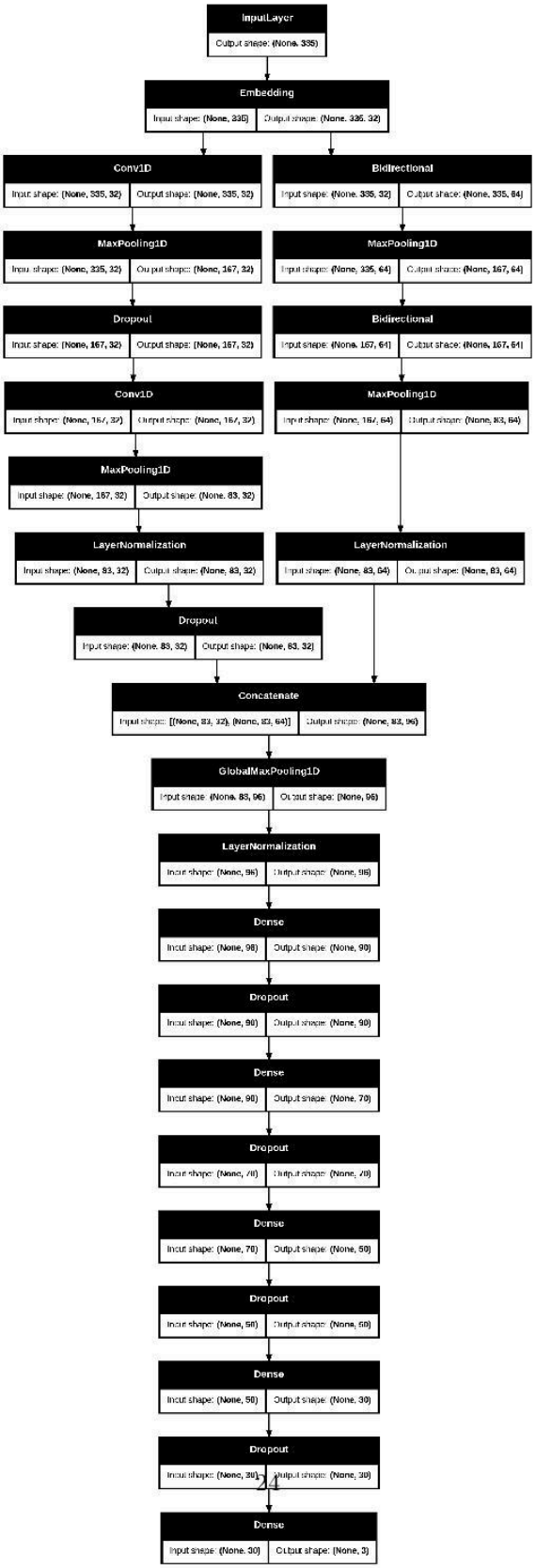
```
max_pooling1d_1 (None, 83, 32) 0 ␣
↪conv1d_1[0][0]
(MaxPooling1D) ␣
↪

bidirectional_1 (None, 167, 64) 18,816 ␣
↪max_pooling1d_2[0][0]
(Bidirectional) ␣
↪

layer_normalization (None, 83, 32) 64 ␣
↪max_pooling1d_1[0][0]
(LayerNormalization) ␣
↪

max_pooling1d_3 (None, 83, 64) 0 ␣
↪bidirectional_1[0][0]
(MaxPooling1D) ␣
↪

dropout_1 (Dropout) (None, 83, 32) 0 ␣
↪layer_normalization[0…

layer_normalization_1 (None, 83, 64) 128 ␣
↪max_pooling1d_3[0][0]
(LayerNormalization) ␣
↪

concatenate (Concatenate) (None, 83, 96) 0 ␣
↪dropout_1[0][0],
␣
↪layer_normalization_1…

global_max_pooling1d (None, 96) 0 ␣
↪concatenate[0][0]
(GlobalMaxPooling1D) ␣
↪

layer_normalization_2 (None, 96) 192 ␣
↪global_max_pooling1d[…
(LayerNormalization) ␣
↪

dense (Dense) (None, 90) 8,730 ␣
↪layer_normalization_2…
```

```
dropout_2 (Dropout)          (None, 90)                0 ␣
↪dense[0][0]

dense_1 (Dense)              (None, 70)            6,370 ␣
↪dropout_2[0][0]

dropout_3 (Dropout)          (None, 70)                0 ␣
↪dense_1[0][0]

dense_2 (Dense)              (None, 50)            3,550 ␣
↪dropout_3[0][0]

dropout_4 (Dropout)          (None, 50)                0 ␣
↪dense_2[0][0]

dense_3 (Dense)              (None, 30)            1,530 ␣
↪dropout_4[0][0]

dropout_5 (Dropout)          (None, 30)                0 ␣
↪dense_3[0][0]

dense_4 (Dense)              (None, 3)                93 ␣
↪dropout_5[0][0]


Total params: 312,369 (1.19 MB)

Trainable params: 312,369 (1.19 MB)

Non-trainable params: 0 (0.00 B)
```

##Visualize the model

[ ]:
```
dot_img_file='model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

[ ]:


InputLayer
Output shape: (None, 335)
Embedding
Input shape: (None, 335) Output shape: (None, 335, 32)
Conv1D
Input shape: (None, 335, 32) Output shape: (None, 335, 32)
Bidirectional
Input shape: (None, 335, 32) Output shape: (None, 335, 64)
MaxPooling1D
Input shape: (None, 335, 32) Output shape: (None, 167, 32)
MaxPooling1D
Input shape: (None, 335, 64) Output shape: (None, 167, 64)
Dropout
Input shape: (None, 167, 32) Output shape: (None, 167, 32)
Bidirectional
Input shape: (None, 167, 64) Output shape: (None, 167, 64)
Conv1D
Input shape: (None, 167, 32) Output shape: (None, 167, 32)
MaxPooling1D
Input shape: (None, 167, 64) Output shape: (None, 83, 64)
MaxPooling1D
Input shape: (None, 167, 32) Output shape: (None, 83, 32)
LayerNormalization
Input shape: (None, 83, 32) Output shape: (None, 83, 32)
LayerNormalization
Input shape: (None, 83, 64) Output shape: (None, 83, 64)
Dropout
Input shape: (None, 83, 32) Output shape: (None, 83, 32)
Concatenate
Input shape: [(None, 83, 32), (None, 83, 64)] Output shape: (None, 83, 96)
GlobalMaxPooling1D
Input shape: (None, 83, 96) Output shape: (None, 96)
LayerNormalization
Input shape: (None, 96) Output shape: (None, 96)
Dense
Input shape: (None, 96) Output shape: (None, 90)
Dropout
Input shape: (None, 90) Output shape: (None, 90)
Dense
Input shape: (None, 90) Output shape: (None, 70)
Dropout
Input shape: (None, 70) Output shape: (None, 70)
Dense
Input shape: (None, 70) Output shape: (None, 50)
Dropout
Input shape: (None, 50) Output shape: (None, 50)
Dense
Input shape: (None, 50) Output shape: (None, 30)
Dropout
Input shape: (None, 30) Output shape: (None, 30)
Dense
Input shape: (None, 30) Output shape: (None, 3)

##Define model checkpoint and training

```
[ ]: callback_model = tf.keras.callbacks.ModelCheckpoint('model_cnn_bilstm.keras',␣
      ↪monitor='val_loss')
     early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
      ↪patience=5, restore_best_weights=True)
     history = model.fit(x=x_train, y=y_train,validation_data=(x_val,␣
      ↪y_val),epochs=10,batch_size=128,callbacks=[callback_model, early_stopping])
```

```
Epoch 1/10
177/177             173s 860ms/step -
accuracy: 0.5936 - loss: 0.9410 - val_accuracy: 0.7572 - val_loss: 0.6055
Epoch 2/10
177/177             210s 906ms/step -
accuracy: 0.7664 - loss: 0.5952 - val_accuracy: 0.7757 - val_loss: 0.5615
Epoch 3/10
177/177             149s 837ms/step -
accuracy: 0.7997 - loss: 0.5175 - val_accuracy: 0.7772 - val_loss: 0.5429
Epoch 4/10
177/177             151s 855ms/step -
accuracy: 0.8091 - loss: 0.4740 - val_accuracy: 0.7775 - val_loss: 0.5508
Epoch 5/10
177/177             199s 841ms/step -
accuracy: 0.8153 - loss: 0.4565 - val_accuracy: 0.7818 - val_loss: 0.5432
Epoch 6/10
177/177             203s 846ms/step -
accuracy: 0.8278 - loss: 0.4361 - val_accuracy: 0.7705 - val_loss: 0.5718
Epoch 7/10
177/177             203s 853ms/step -
accuracy: 0.8408 - loss: 0.4086 - val_accuracy: 0.7745 - val_loss: 0.5591
Epoch 8/10
177/177             202s 855ms/step -
accuracy: 0.8453 - loss: 0.3932 - val_accuracy: 0.7756 - val_loss: 0.5646
```

##Evaluate the prediction results on the test set and print a detailed performance report.

```
[ ]: y_pred = model.predict(x_test)
     y_pred_classes = y_pred.argmax(axis=1)
     y_test_classes = y_test.argmax(axis=1)
     report = classification_report(y_test_classes, y_pred_classes)
     print(report)
```

```
79/79             10s 108ms/step
              precision    recall  f1-score   support

           0       0.65      0.88      0.75       544
```

```
           1       0.00      0.00      0.00       366
           2       0.85      0.95      0.90      1607

    accuracy                           0.79      2517
   macro avg       0.50      0.61      0.55      2517
weighted avg       0.69      0.79      0.73      2517
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

#Inference Model with input test

[ ]:
```
def preprocess_raw_input(raw_input, tokenizer):
    input_text_pre = list(tf.keras.preprocessing.text.
 ↪text_to_word_sequence(raw_input))
    input_text_pre = " ".join(input_text_pre)
    input_text_pre_accent = ViTokenizer.tokenize(input_text_pre)
    print("Text preprocessed: ", input_text_pre_accent)
    tokenized_data_text = tokenizer.texts_to_sequences([input_text_pre_accent])
    vec_data = pad_sequences(tokenized_data_text, padding='post', maxlen=335)
    return vec_data

def inference_model(input_feature, model):
  output = model(input_feature).numpy()[0]
  result = output.argmax()
  conf = float(output.max())
  label_dict = {'tiêu cực':0, 'trung lập':1, 'tích cực':2}
  label = list(label_dict.keys())
  return label[int(result)], conf

def prediction(raw_input, tokenizer, model):
    input_model = preprocess_raw_input(raw_input, tokenizer)
    result, conf = inference_model(input_model, model)
    return result, conf
```

```
my_model = generate_model()
my_model = load_model('model_cnn_bilstm.keras')
with open(r"tokenizer_data.pkl", "rb") as input_file:
    my_tokenizer = pickle.load(input_file)

print(prediction("sữa chua ngon lắm",my_tokenizer,my_model))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(

Text preprocessed:  sữa_chua ngon lắm
('tích cực', 0.9667492508888245)
```

[ ]:
```
while(True):
  text = input()
  if text == "end":
    break
  else:
    # Convert the prediction result to a string using str()
    print(str(prediction(text,my_tokenizer,my_model)[0])+"\n")
```

```
print(prediction("sữa chua ngon lắm",my_tokenizer,my_model))
Text preprocessed:  print prediction sữa_chua ngon lắm my tokenizer my model
('tích cực', 0.9524750709533691)
Các vị có thể cho tại hạ xin link
Text preprocessed:  các vị có_thể cho tại hạ xin link
('trung lập', 0.46215373277664185)
end
```

[ ]:
```
# import pickle
# from tensorflow.keras.models import load_model

# # Tải mô hình từ file .keras
# model = load_model("model_cnn_bilstm.keras")

# # Lưu mô hình bằng pickle
# with open("model.pkl", "wb") as f:
#     pickle.dump(model, f)
```

[ ]:
```
import pickle
from google.colab import files


# Download the model and tokenizer files
#files.download('model_cnn_bilstm.keras')
```

2.3 Deploy model

2.3.1 Setting up data source

Trong thư mục Voz_neww/Voz_neww/spiders. File demospiders.py sẽ làm nhiệm vụ trọng yếu là cào hết tất cả các dữ liệu trong khoảng 10 phút so với thời gian thực từ trang web voz.vn. Các dữ liệu thu vào được bao gồm: mã số id, chủ đề thread, ngày đăng thread, người comment mới nhất, thời gian đăng comment mới nhất, tin nhắn và đường dẫn tới tin nhắn đó

```
import scrapy
import asyncio
from scrapy.spiders import Spider
from urllib.parse import urljoin, urlparse
from datetime import datetime, timedelta, timezone
import hashlib
import time
import re
class DemospiderSpider(Spider):
    name = "demospider"
    allowed_domains = ["voz.vn"]
    start_urls = ["https://voz.vn/whats-new"]

    processed_posts = set()
    last_scraped_timestamp = None

    custom_settings = {
        'CONCURRENT_REQUESTS': 1,
        'DOWNLOAD_DELAY': 1,
        'COOKIES_ENABLED': True,
        'ROBOTSTXT_OBEY': True,
        'DUPEFILTER_CLASS': 'scrapy.dupefilters.BaseDupeFilter'
    }

    def extract_thread_id(self, url):
        parsed = urlparse(url)
        path_parts = parsed.path.split('.')
```

```
        if len(path_parts) > 1:
            return path_parts[-1]
        return None

    def generate_item_id(self, thread_url, timestamp):
        thread_id = self.extract_thread_id(thread_url)
        if thread_id and timestamp:
            id_string = f"{thread_id}_{timestamp}"
            return hashlib.md5(id_string.encode()).hexdigest()
        return None

    def start_requests(self):
        while True:
            yield scrapy.Request(
                self.start_urls[0],
                callback=self.parse,
                dont_filter=True,
                meta={'dont_cache': True}
            )
            time.sleep(10)

    def parse(self, response):
        thread_containers = response.xpath("//div[contains(@class, 'structItem structItem--thread')]")

        threads = []
        for thread in thread_containers:
            latest_link = thread.xpath(".//div[@class='structItem-cell
structItem-cell--latest']//a[contains(@href, '/latest')]/@href").get()
            if latest_link:
                thread_url = urljoin(response.url, latest_link)
                thread_date_str = thread.xpath(".//div[@class='structItem-cell
structItem-cell--main']//time/@datetime").get()

                thread_info = {
                    'url': thread_url,
                    'thread_title': thread.xpath(".//div[@class='structItem-title']//a/text()").get(),
                    'thread_date': thread_date_str,
                    'timestamp': datetime.fromisoformat(thread_date_str.replace('Z', '+00:00')) if
thread_date_str else None
                }
```

```
            threads.append(thread_info)

        sorted_threads = sorted(threads, key=lambda x: x['timestamp'] if x['timestamp'] else
datetime.min)

        for thread in sorted_threads:
            yield scrapy.Request(
                thread['url'],
                callback=self.parse_latest_message,
                meta={'thread_info': thread},
                dont_filter=True
            )

    async def parse_latest_message(self, response):
        thread_info = response.meta['thread_info']

        message_containers = response.xpath("//article[contains(@class, 'message message--post')]")
        sorted_messages = sorted(
            message_containers,
            key=lambda m: m.xpath(".//time[@class='u-dt']/@datetime").get()
        )

        # Calculate the cutoff time: current time (UTC) - 10 minutes
        time_threshold = datetime.utcnow().replace(tzinfo=timezone.utc) - timedelta(minutes=10)

        new_messages = []
        for message_container in sorted_messages:
            message_content = message_container.xpath(".//div[contains(@class,
'message-userContent')]//div[@class='bbWrapper']//text()[not(ancestor::blockquote)]").getall()
            message_content = ' '.join([text.strip() for text in message_content if text.strip()])
            pattern = r'\{\n\t+\"lightbox_.*?\"Toggle sidebar\"\n\t+\}'

            message_content = re.sub(pattern, '', message_content, flags=re.DOTALL).strip()
            username =
message_container.xpath(".//h4[@class='message-name']//span[@itemprop='name']/text()").get()
            timestamp_str = message_container.xpath(".//time[@class='u-dt']/@datetime").get()
            timestamp = datetime.fromisoformat(timestamp_str.replace('Z', '+00:00')) if timestamp_str
else None

            if timestamp and timestamp >= time_threshold:
```

```
            item_id = self.generate_item_id(response.url, timestamp_str)
            if item_id and item_id not in self.processed_posts:
                new_messages.append({
                    'id': item_id,
                    'thread_title': thread_info['thread_title'],
                    'thread_date': thread_info['thread_date'],
                    'latest_poster': username,
                    'latest_post_time': timestamp_str,
                    'message_content': message_content,
                    'thread_url': response.url
                })

        # Yield each message with a delay between them
        for message in new_messages:
            self.processed_posts.add(message['id'])
            self.logger.info(f"Yielding post: {message['thread_title']} from user {message['latest_poster']}")
            yield message
            await asyncio.sleep(1)  # Set a 1-second delay between each message

        if not new_messages:
            self.logger.info("No new messages found. Waiting 10 seconds before checking again.")
            await asyncio.sleep(10)  # Wait 10 seconds if no new messages are found

        # Resume from the last scraped timestamp in future requests
        if self.last_scraped_timestamp:
            self.logger.info(f"Resuming from last timestamp: {self.last_scraped_timestamp}")
```

Sau khi extract được, từng data đã extract sẽ tiến hành đi vào phần xử lý chính là pipeline.py

2.3.2 Transform and load

2.3.2.1 Transform

Sau khi đã extract. Pipeline.py sẽ có nhiệm vụ biến đổi những data cần thiết. Class FetchMessagePipeline dưới đây sẽ sử dụng các model đã được train trước đó để làm nhiệm vụ dự đoán cảm xúc cho từng “message content” và cập nhật data. Các hàm preprocess_raw_input , inference_model, prediction dùng để gọi model và dự đoán cảm xúc

```
import psycopg2
import logging
from datetime import datetime
import pytz
from tensorflow.keras.models import load_model
import pickle
import tensorflow as tf
from pyvi import ViTokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import model_from_json
tz = pytz.timezone('Asia/Ho_Chi_Minh')
# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def preprocess_raw_input(raw_input, tokenizer):
    input_text_pre = list(tf.keras.preprocessing.text.text_to_word_sequence(raw_input))
    input_text_pre = " ".join(input_text_pre)
    input_text_pre_accent = ViTokenizer.tokenize(input_text_pre)
    tokenized_data_text = tokenizer.texts_to_sequences([input_text_pre_accent])
    vec_data = pad_sequences(tokenized_data_text, padding='post', maxlen=335)
    return vec_data

def inference_model(input_feature, model):
    output = model(input_feature).numpy()[0]
    result = output.argmax()
    label_dict = {'tiêu cực': 0, 'trung lập': 1, 'tích cực': 2}
    label = list(label_dict.keys())
    return label[int(result)]
```

```
def prediction(raw_input, tokenizer, model):
    input_model = preprocess_raw_input(raw_input, tokenizer)
    result= inference_model(input_model, model)
    return result
model_path = "./models/model.pkl"
tokenizer_data_path = "./models/tokenizer_data.pkl"
with open(model_path, "rb") as model_file:
    model = pickle.load(model_file)

with open(tokenizer_data_path, "rb") as tokenizer_file:
    my_tokenizer = pickle.load(tokenizer_file)

class FetchMessagePipeline:

    def analyze_sentiment(self, text):
        try:
            # Get sentiment label from underthesea
            sentiment_label = prediction(text,my_tokenizer,model)

            # Convert sentiment labels to counts
            sentiment_counts = {
                'positive': 0,
                'negative': 0,
                'neutral': 0
            }

            # Increment the appropriate counter based on sentiment
            if sentiment_label == 'tích cực':
                sentiment_counts['positive'] = 1
            elif sentiment_label == 'tiêu cực':
                sentiment_counts['negative'] = 1
            else:
                sentiment_counts['neutral'] = 1

            return sentiment_counts

        except Exception as e:
            logger.error(f"Error in sentiment analysis: {str(e)}")
            return {'positive': 0, 'negative': 0, 'neutral': 0}
```

```
def process_item(self, item, spider):
    """Process each scraped item"""
    try:
        # Get the message content
        message_text = item['message_content']

        # Analyze sentiment
        sentiment_counts = self.analyze_sentiment(message_text)

        # Update the item with sentiment counts
        item.update({
            **sentiment_counts,
            'processed_at': datetime.now(tz).isoformat()
        })

        logger.info(f"Processed item {item['id']}")
        return item

    except Exception as e:
        logger.error(f"Error processing item: {str(e)}")
        return item
```

2.3.2.2 Load

Sau khi đã xong transform xong data sẽ có class tiến SentimentAnalysisPipeline tiến hành gọi database và load dữ liệu vào đóng quá trình scrap cho mỗi data đầu vào

```
class SentimentAnalysisPipeline:
    def __init__(self):
        try:
            self.conn = psycopg2.connect(
                dbname="vozdb",
                user="postgres",
                password="postgres",
                host="db",
            )
            self.cur = self.conn.cursor()
```

```
        except Exception as e:
            logger.error(f"Error connecting to database: {str(e)}")

    def process_item(self, item, spider):
        try:
            # Store in database with sentiment counts
            self.cur.execute("""
                INSERT INTO voz_messages (
                    id, thread_title, thread_date, latest_poster,
                    latest_post_time, message_content, thread_url,
                    positive_count, negative_count, neutral_count,
                    analyzed_at
                ) VALUES (
                    %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
                )
                ON CONFLICT (id) DO UPDATE SET
                    positive_count = EXCLUDED.positive_count,
                    negative_count = EXCLUDED.negative_count,
                    neutral_count = EXCLUDED.neutral_count,
                    analyzed_at = EXCLUDED.analyzed_at
            """, (
                item['id'], item['thread_title'], item['thread_date'],
                item['latest_poster'], item['latest_post_time'],
                item['message_content'], item['thread_url'],
                item['positive'], item['negative'], item['neutral'],
                item['processed_at']
            ))

            self.conn.commit()
            logger.info(f"Successfully stored item {item['id']} in database")

        except Exception as e:
            logger.error(f"Error storing item {item['id']} in database: {str(e)}")
            self.conn.rollback()

        return item

    def close_spider(self, spider):
        self.cur.close()
        self.conn.close()
```

2.3.3 Backend

Sau khi đã load xong dữ liệu vào database tiến hành tạo API bằng FastAPI.Các chức năng của Hàm sau đây:

- wait_for_db() : Gọi database, nếu không có database sẽ chờ 2 giây và gọi lại, cho tới lần thứ 30 sẽ trả về lỗi nếu không kết nối được
- get_db_connection(): Tiến hành kết nối với database dựa trên config
- get_sentiment_stats(conn, limit=15): Lấy thống kê cảm xúc trong khoảng thời gian nhất định (số giây gần nhất) từ cơ sở dữ liệu.
- get_sentiment_summary(conn): Lấy tóm tắt tổng thể về cảm xúc trong vòng 24 giờ qua
- get_messages_with_sentiment: Lấy full tổng thể của dữ liệu đầu ra
- Cac api endpoint

```python
# api/main.py
from datetime import datetime, timedelta
from fastapi import FastAPI, WebSocket, WebSocketDisconnect, Depends, Query, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import psycopg2
from psycopg2.extras import RealDictCursor
import os
import logging
import time
from contextlib import contextmanager
from typing import Optional, List

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
```

```
logger = logging.getLogger(__name__)

# Initialize FastAPI app
app = FastAPI(title="VOZ Analytics API")
origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Database configuration
DB_CONFIG = {
    "dbname": "vozdb",
    "user": "postgres",
    "password": "postgres",
    "host": "db",
    "port": "5432"
}

def wait_for_db(max_retries=30, delay_seconds=2):
    """Wait for database to be ready"""
    retries = 0
    while retries < max_retries:
        try:
            conn = psycopg2.connect(**DB_CONFIG)
            conn.close()
            logger.info("Successfully connected to the database")
            return True
        except psycopg2.Error as e:
            retries += 1
            logger.warning(f"Attempt {retries}/{max_retries} to connect to database failed: {str(e)}")
            logger.warning("Retrying in %s seconds...", delay_seconds)
            time.sleep(delay_seconds)

    raise Exception("Could not connect to the database after multiple attempts")
```

```python
@contextmanager
def get_db_connection():
    """Context manager for database connections"""
    conn = None
    try:
        conn = psycopg2.connect(**DB_CONFIG, cursor_factory=RealDictCursor)
        yield conn
    except psycopg2.Error as e:
        logger.error(f"Database connection error: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database connection error: {str(e)}")
    finally:
        if conn:
            conn.close()
            logger.debug("Database connection closed")

def get_db():
    """Database dependency for FastAPI"""
    with get_db_connection() as conn:
        yield conn

# Analytics queries
def get_sentiment_stats(conn, limit=15):
    """Get the last `limit` seconds of sentiment statistics"""
    try:
        with conn.cursor() as cur:
            query = f"""
            WITH RECURSIVE time_series AS (
                SELECT NOW() - INTERVAL '1 second' * generate_series(0, {limit - 1}) AS check_time
            )
            SELECT
                ts.check_time AS check_time,
                (SELECT SUM(positive_count) FROM voz_messages) AS total_positive_count,
                (SELECT SUM(negative_count) FROM voz_messages) AS total_negative_count,
                (SELECT SUM(neutral_count) FROM voz_messages) AS total_neutral_count,
                (SELECT COUNT(*) FROM voz_messages) AS total_messages
            FROM time_series ts
            ORDER BY ts.check_time DESC;
            """
            cur.execute(query)
            results = cur.fetchall()
```

```
        return results
    except psycopg2.Error as e:
        logger.error(f"Error fetching sentiment stats list: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")


def get_sentiment_summary(conn):
    """Get overall sentiment summary for the last 24 hours"""
    try:
        with conn.cursor() as cur:
            query = """
                SELECT
                    SUM(positive_count) as total_positive,
                    SUM(negative_count) as total_negative,
                    SUM(neutral_count) as total_neutral,
                    COUNT(*) as total_messages
                FROM voz_messages
                WHERE analyzed_at >= NOW() - INTERVAL '24 hours'
            """
            cur.execute(query)
            result = cur.fetchone()
            return result
    except psycopg2.Error as e:
        logger.error(f"Error fetching sentiment summary: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")

def get_messages_with_sentiment(conn, limit: int = 10, offset: int = 0, thread_id: Optional[str] =
None):
    """Get messages with their sentiment analysis"""
    try:
        with conn.cursor() as cur:
            query = """
                SELECT
                    id,
                    thread_title,
                    thread_date,
                    message_content,
                    latest_poster,
                    latest_post_time,
                    thread_url,
```

```
            CASE
                WHEN positive_count = 1 THEN 'positive'
                WHEN negative_count = 1 THEN 'negative'
                ELSE 'neutral'
            END as sentiment,
            analyzed_at
        FROM voz_messages
        WHERE 1=1
    """
    params = []

    if thread_id:
        query += " AND thread_id = %s"
        params.append(thread_id)

    query += """
        ORDER BY analyzed_at DESC
        LIMIT %s OFFSET %s
    """
    params.extend([limit, offset])

    cur.execute(query, params)
    messages = cur.fetchall()

    # Get total count for pagination
    count_query = """
        SELECT COUNT(*) as total
        FROM voz_messages
        WHERE 1=1
    """
    if thread_id:
        count_query += " AND thread_id = %s"
        cur.execute(count_query, [thread_id] if thread_id else None)
    else:
        cur.execute(count_query)

    total_count = cur.fetchone()['total']

    return {
        "messages": messages,
```

```
            "total": total_count,
            "limit": limit,
            "offset": offset
        }
    except psycopg2.Error as e:
        logger.error(f"Error fetching messages with sentiment: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Database query error: {str(e)}")


# API endpoints
@app.get("/stats/sentiment/iter")
@app.get("/stats/sentiment/iter")
def sentiment_stats(
    conn = Depends(get_db),
    limit: int = Query(15, ge=1, le=60)  # Limit can be adjusted (default 15, max 60)
):
    """Get the last `limit` seconds of sentiment statistics"""
    return get_sentiment_stats(conn, limit=limit)

@app.get("/stats/sentiment/summary")
def sentiment_summary(conn = Depends(get_db)):
    """Get overall sentiment summary"""
    return get_sentiment_summary(conn)

@app.get("/messages/sentiment")
def get_messages(
    conn = Depends(get_db),
    limit: int = Query(5, ge=1, le=100),
    offset: int = Query(0, ge=0),
    thread_id: Optional[str] = None
):
    """Get messages with their sentiment analysis"""
    return get_messages_with_sentiment(conn, limit, offset, thread_id)

# Health check endpoint
@app.get("/health")
async def health_check():
    """Health check endpoint that also verifies database connection"""
    try:
        with get_db_connection() as conn:
```

```
        with conn.cursor() as cur:
            cur.execute("SELECT 1")
            return {
                "status": "healthy",
                "database": "connected",
                "timestamp": datetime.now().isoformat()
            }
    except Exception as e:
        return {
            "status": "unhealthy",
            "database": "disconnected",
            "error": str(e),
            "timestamp": datetime.now().isoformat()
        }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True, log_level="info")
```

2.3.4 Frontend

Ở phần này chỉ chú ý đến phần trong file src của vite-project. Trong src sẽ có phần chính là App.jsx để tiến hành chạy frontend. Tụi em sẽ sử dụng thư viện của MaterialUI (MUI) để hình hoá dữ liệu.

```
import { useState, useEffect } from 'react';
import './App.css';
import { PieChart } from '@mui/x-charts/PieChart';
import { LineChart } from '@mui/x-charts/LineChart';
import {
 Typography,
 List,
 ListItem,
 Card,
 CardContent,
 CardHeader,
 Divider,
 Box,
```

```
 Paper,
 Link,
} from '@mui/material';

const API_ENDPOINTS = {
 sentimentSummary: 'http://localhost:8000/stats/sentiment/summary',
 sentimentMessages: 'http://localhost:8000/messages/sentiment',
 sentimentByTime: 'http://localhost:8000/stats/sentiment/iter',
};
function formatSentimentData(data) {
 const totalMessages = data.total_messages ?? 0;
 const positive = totalMessages ? ((data.total_positive / totalMessages) * 100).toFixed(2) : 0;
 const negative = totalMessages ? ((data.total_negative / totalMessages) * 100).toFixed(2) : 0;
 const neutral = totalMessages ? ((data.total_neutral / totalMessages) * 100).toFixed(2) : 0;
 return {
  chartData: [
   { id: 0, value: data.total_positive ?? 0, label: 'Positive' },
   { id: 1, value: data.total_negative ?? 0, label: 'Negative' },
   { id: 2, value: data.total_neutral ?? 0, label: 'Neutral' },
  ],
  summary: {
   totalMessages,
   positive,
   negative,
   neutral,
  },
 };
}
function normalizeSentimentData(dataArray) {
 return dataArray.map(data => {
  const hourDate = new Date(data.check_time);
  const total = data.total_messages;
  return {
   Hour: hourDate,
   positive: total ? Number((data.total_positive_count / total * 100).toFixed(2)) : 0,
   negative: total ? Number((data.total_negative_count / total * 100).toFixed(2)) : 0,
   neutral: total ? Number((data.total_neutral_count / total * 100).toFixed(2)) : 0,
  };
 });
}
function App() {
 const [sentiment, setSentiment] = useState([]);
 const [sentimentSummary, setSentimentSummary] = useState({});
```

```
const [messages, setMessages] = useState([]);
const [analysisByTime, setAnalysisByTime] = useState([]);

const fetchData = async (endpoint, processData) => {
  try {
    const response = await fetch(endpoint);
    if (!response.ok) {
      throw new Error(`Failed to fetch: ${endpoint}`);
    }
    const data = await response.json();
    processData(data);
  } catch (error) {
    console.error(`Error fetching data from ${endpoint}:`, error);
  }
};

useEffect(() => {
  const fetchAllData = () => {
    fetchData(API_ENDPOINTS.sentimentSummary, data => {
      const formatted = formatSentimentData(data);
      setSentiment(formatted.chartData);
      setSentimentSummary(formatted.summary);
    });
    fetchData(API_ENDPOINTS.sentimentMessages, data => setMessages(data.messages));
    fetchData(API_ENDPOINTS.sentimentByTime, data => {
      const normalizedData = normalizeSentimentData(data);
      setAnalysisByTime(normalizedData);
    });
  };

  fetchAllData();
  const intervalId = setInterval(fetchAllData, 1000);

  return () => clearInterval(intervalId);
}, []);

return (
  <Box sx={{ padding: '2rem', backgroundColor: '#f9f9f9', minHeight: '100vh' }}>
    {/* Summary Boxes */}
    <Box
      sx={{
        display: 'flex',
        gap: '1.5rem',
```

```
    marginBottom: '2rem',
    justifyContent: 'space-between',
  }}
>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Total Messages</Typography>
    <Typography variant="h4">{sentimentSummary.totalMessages || 0}</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Positive (%)</Typography>
    <Typography variant="h4">{sentimentSummary.positive || 0}%</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Negative (%)</Typography>
    <Typography variant="h4">{sentimentSummary.negative || 0}%</Typography>
  </Paper>
  <Paper elevation={3} sx={{ padding: '1rem', flex: 1, textAlign: 'center' }}>
    <Typography variant="h6">Neutral (%)</Typography>
    <Typography variant="h4">{sentimentSummary.neutral || 0}%</Typography>
  </Paper>
</Box>

{/* Main Content */}
<Box sx={{ display: 'flex', gap: '2rem' }}>
  {/* Left Section */}
  <Box sx={{ flex: 7 }}>
    <Card elevation={3} sx={{ marginBottom: '2rem' }}>
      <CardHeader title="Sentiment Distribution" />
      <Divider />
      <CardContent>
        {sentiment.length > 0 ? (
          <PieChart
            series={[{ data: sentiment }]}
            width={600}
            height={400}
          />
        ) : (
          <Typography>Loading sentiment data...</Typography>
        )}
      </CardContent>
    </Card>

    <Card elevation={3}>
```

```
    <CardHeader title="Sentiment Over Time" />
    <Divider />
    <CardContent>
     {analysisByTime.length > 0 ? (
      <LineChart
       height={400}
       series={[
        {
         data: analysisByTime.map(item => item.positive),
         label: 'Positive',
         area: true,
         stack: 'total',
         showMark: false,
        },
        {
         data: analysisByTime.map(item => item.negative),
         label: 'Negative',
         area: true,
         stack: 'total',
         showMark: false,
        },
        {
         data: analysisByTime.map(item => item.neutral),
         label: 'Neutral',
         area: true,
         stack: 'total',
         showMark: false,
        },
       ]}
       xAxis={[
        {
         data: analysisByTime.map(item => item.Hour),
         scaleType: 'time',
         valueFormatter: item => item.toLocaleTimeString(),
        },
       ]}
      />
     ) : (
      <Typography>Loading time-series data...</Typography>
     )}
    </CardContent>
   </Card>
  </Box>
```

```
    {/* Right Section */}
    <Box sx={{ flex: 3 }}>
      <Card elevation={3}>
        <CardHeader title="Recent Messages" />
        <Divider />
        <CardContent>
          <List>
            {messages.length > 0 ? (
              messages.map(item => (
                <ListItem key={item.id} sx={{ marginBottom: '1rem' }}>
                  <Paper elevation={1} sx={{ padding: '1rem', width: '100%' }}>
                    <Typography variant="body1">
                      <strong>Time:</strong> {new Date(item.latest_post_time).toLocaleString()}
                    </Typography>
                    <Typography variant="body1">
                      <strong>Message:</strong> {item.message_content}
                    </Typography>
                    <Typography variant="body1">
                      <strong>Thread:</strong>{' '}
                      <Link href={item.thread_url} target="_blank" rel="noopener noreferrer">
                        {item.thread_url}
                      </Link>
                    </Typography>
                    <Typography variant="body1">
                      <strong>Sentiment:</strong> {item.sentiment}
                    </Typography>
                  </Paper>
                </ListItem>
              ))
            ) : (
              <Typography>No messages to display.</Typography>
            )}
          </List>
        </CardContent>
      </Card>
    </Box>
  </Box>
);
}
export default App;
```

2.3.5 Docker build file

Để dự án này có thể chạy đơn giản không cần phải tải môi trường hay cài bất cứ phần mềm chạy code nào, chúng em sẽ sử dụng Docker làm việc đó. Và sau đây là các file cần thiết để build cho dự án này :

- Docker-compose.yaml: Dùng để xây dựng các image cần thiết cho dự án. Trong đó có các image như sau :
  - db: Xây dựng image cho csdl postgresql và thực thi lệnh sql trong file init.sql
  - api: Xây dựng image cho backend và khởi tạo chạy FastAPI dựa trên file Dockerfile trong Voz_neww
  - scrapy: Dựa vào image của backend và bắt đầu scrap data về
  - frontend: Xây dựng image cho frontend dựa trên Dockerfile trong vite_project

```yaml
services:
 db:
  image: postgres:13
  volumes:
   - ./VOZ_neww/postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
  environment:
   POSTGRES_USER: postgres
   POSTGRES_PASSWORD: postgres
   POSTGRES_DB: vozdb
  ports:
   - "5433:5432"
  healthcheck:
   test: ["CMD-SHELL", "pg_isready -U postgres"]
   interval: 5s
   timeout: 5s
   retries: 5
  networks:
   - voz_dash

 api:
  build: ./VOZ_neww
```

```
  command: ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
  ports:
    - "8000:8000"
  volumes:
    - ./VOZ_neww:/app
  depends_on:
    db:
      condition: service_healthy
  environment:
    DATABASE_URL: postgresql://postgres:postgres@db:5432/vozdb
  networks:
    - voz_dash

scrapy:
  build: ./VOZ_neww
  command: ["scrapy", "crawl", "demospider"]
  volumes:
    - ./VOZ_neww:/app
  depends_on:
    db:
      condition: service_healthy
  networks:
    - voz_dash

frontend:
  build: ./vite-project
  ports:
    - "5173:5173"
  networks:
    - voz_dash


networks:
voz_dash:
  driver: bridge
```

Vite-project/Dockerfile:

```
FROM node:18.17.1
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

Voz_new/Dockerfile

```
FROM python:3.11

WORKDIR /app

RUN apt-get update && apt-get install -y \
  postgresql-client \
  cron \
  && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY models/ /app/models/

COPY start.sh /app/start.sh
RUN chmod +x /app/start.sh
EXPOSE 8000
```

SETUP A DEMO

1. Ở bước này, để có thể khởi tạo project, trước hết hãy tải docker về máy tính.
2. Sau đó vào đường link github https://github.com/Amature123/new_new_sentiment và clone project về
3. Bật màn command promt hoặc terminal và đi tới địa chỉ của project
4. Gõ Docker compose build để tiến hành tải các image cần thiết
5. Sau khi đã tải các project cần thiết thì tiến hành gõ docker compose up để chạy code
6. Sau khi đã hoàn tất hãy vào localhost:5183 và sẽ hiện hết các dashboard.

Tài liệu tham khảo

- **Docker:** https://www.docker.com/
- **Frontend:** https://vite.dev/
- **Backend:** https://fastapi.tiangolo.com/
- Model: https://www.tensorflow.org/
- Scrapy: https://scrapy.org/
  Project được lấy ví dụ từ
- https://github.com/nama1arpit/reddit-streaming-pipeline
- https://github.com/jalvin99/RedditSentimentAnalyzer
- https://github.com/undertheseanlp/underthesea