```python
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import yaml

import logging

# logging configure

logger = logging.getLogger('data_ingestion')
logger.setLevel('DEBUG')

console_handler = logging.StreamHandler()
console_handler.setLevel('DEBUG')

file_handler = logging.FileHandler('errors.log')
file_handler.setLevel('ERROR')

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
- %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)

logger.addHandler(console_handler)
logger.addHandler(file_handler)

def load_params(params_path: str) -> float:
    try:
        with open(params_path, 'r') as file:
            params = yaml.safe_load(file)
        test_size = params['data_ingestion']['test_size']
        logger.debug('test size retrieved')
        return test_size
    except FileNotFoundError:
        logger.error('File not found')
        raise
    except yaml.YAMLError as e:
        logger.error('yaml error')
        raise
    except Exception as e:
        logger.error('some error occured')
        raise

def load_data(data_url: str) -> pd.DataFrame:
    try:
        df = pd.read_csv(data_url)
        return df
    except pd.errors.ParserError as e:
        print(f"Error: Failed to parse the CSV file from {data_url}.")
```

```
        print(e)
        raise
    except Exception as e:
        print(f"Error: An unexpected error occurred while loading the
data.")
        print(e)
        raise

def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
    try:
        df.drop(columns=['tweet_id'], inplace=True)
        final_df = df[df['sentiment'].isin(['happiness', 'sadness'])]
        final_df['sentiment'].replace({'happiness': 1, 'sadness': 0},
inplace=True)
        return final_df
    except KeyError as e:
        print(f"Error: Missing column {e} in the dataframe.")
        raise
    except Exception as e:
        print(f"Error: An unexpected error occurred during
preprocessing.")
        print(e)
        raise

def save_data(train_data: pd.DataFrame, test_data: pd.DataFrame,
data_path: str) -> None:
    try:
        data_path = os.path.join(data_path, 'raw')
        os.makedirs(data_path, exist_ok=True)
        train_data.to_csv(os.path.join(data_path, "train.csv"),
index=False)
        test_data.to_csv(os.path.join(data_path, "test.csv"),
index=False)
    except Exception as e:
        print(f"Error: An unexpected error occurred while saving the
data.")
        print(e)
        raise

def main():
    try:
        test_size = load_params(params_path='params1.yaml')
        df =
load_data(data_url='https://raw.githubusercontent.com/campusx-
official/jupyter-masterclass/main/tweet_emotions.csv')
        final_df = preprocess_data(df)
        train_data, test_data = train_test_split(final_df,
test_size=test_size, random_state=42)
        save_data(train_data, test_data, data_path='data')
    except Exception as e:
```

```
        print(f"Error: {e}")
        print("Failed to complete the data ingestion process.")

if __name__ == '__main__':
    main()

import numpy as np
import pandas as pd

import os

import re
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer, WordNetLemmatizer

# fetch the data from data/raw
train_data = pd.read_csv('./data/raw/train.csv')
test_data = pd.read_csv('./data/raw/test.csv')

# transform the data
nltk.download('wordnet')
nltk.download('stopwords')

def lemmatization(text):
    lemmatizer= WordNetLemmatizer()

    text = text.split()

    text=[lemmatizer.lemmatize(y) for y in text]

    return " " .join(text)

def remove_stop_words(text):
    stop_words = set(stopwords.words("english"))
    Text=[i for i in str(text).split() if i not in stop_words]
    return " ".join(Text)

def removing_numbers(text):
    text=''.join([i for i in text if not i.isdigit()])
    return text

def lower_case(text):

    text = text.split()

    text=[y.lower() for y in text]

    return " " .join(text)
```

```
def removing_punctuations(text):
    ## Remove punctuations
    text = re.sub('[%s]' % re.escape("""!"#$%&'()*+,،-./:;<=>؟?
@[\]^_`{|}~"""), ' ', text)
    text = text.replace('؛',"", )

    ## remove extra whitespace
    text = re.sub('\s+', ' ', text)
    text =  " ".join(text.split())
    return text.strip()

def removing_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

def remove_small_sentences(df):
    for i in range(len(df)):
        if len(df.text.iloc[i].split()) < 3:
            df.text.iloc[i] = np.nan

def normalize_text(df):
    df.content=df.content.apply(lambda content : lower_case(content))
    df.content=df.content.apply(lambda content :
remove_stop_words(content))
    df.content=df.content.apply(lambda content :
removing_numbers(content))
    df.content=df.content.apply(lambda content :
removing_punctuations(content))
    df.content=df.content.apply(lambda content :
removing_urls(content))
    df.content=df.content.apply(lambda content :
lemmatization(content))
    return df

train_processed_data = normalize_text(train_data)
test_processed_data = normalize_text(test_data)

# store the data inside data/processed
data_path = os.path.join("data","processed")

os.makedirs(data_path)

train_processed_data.to_csv(os.path.join(data_path,"train_processed.cs
v"))
test_processed_data.to_csv(os.path.join(data_path,"test_processed.csv"
))

import numpy as np
import pandas as pd
```

```
import os

from sklearn.feature_extraction.text import CountVectorizer

# fetch the data from data/processed
train_data = pd.read_csv('./data/processed/train_processed.csv')
test_data = pd.read_csv('./data/processed/test_processed.csv')

train_data.fillna('',inplace=True)
test_data.fillna('',inplace=True)

# apply BoW
X_train = train_data['content'].values
y_train = train_data['sentiment'].values

X_test = test_data['content'].values
y_test = test_data['sentiment'].values

# Apply Bag of Words (CountVectorizer)
vectorizer = CountVectorizer(max_features=50)

# Fit the vectorizer on the training data and transform it
X_train_bow = vectorizer.fit_transform(X_train)

# Transform the test data using the same vectorizer
X_test_bow = vectorizer.transform(X_test)

train_df = pd.DataFrame(X_train_bow.toarray())

train_df['label'] = y_train

test_df = pd.DataFrame(X_test_bow.toarray())

test_df['label'] = y_test

# store the data inside data/features
data_path = os.path.join("data","features")

os.makedirs(data_path)

train_df.to_csv(os.path.join(data_path,"train_bow.csv"))
test_df.to_csv(os.path.join(data_path,"test_bow.csv"))


import numpy as np
import pandas as pd
import pickle

from sklearn.ensemble import GradientBoostingClassifier

# fetch the data from data/processed
```

```
train_data = pd.read_csv('./data/features/train_bow.csv')

X_train = train_data.iloc[:,0:-1].values
y_train = train_data.iloc[:,-1].values

# Define and train the XGBoost model

clf = GradientBoostingClassifier(n_estimators=50)
clf.fit(X_train, y_train)

# save
pickle.dump(clf, open('model.pkl','wb'))


import numpy as np
import pandas as pd

import pickle
import json

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score,
roc_auc_score

clf = pickle.load(open('model.pkl','rb'))
test_data = pd.read_csv('./data/features/test_bow.csv')

X_test = test_data.iloc[:,0:-1].values
y_test = test_data.iloc[:,-1].values

y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)[:, 1]

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)

metrics_dict={
    'accuracy':accuracy,
    'precision':precision,
    'recall':recall,
    'auc':auc
}

with open('metrics.json', 'w') as file:
    json.dump(metrics_dict, file, indent=4)
```