```
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
import re


# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

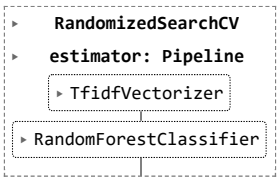
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# Load the dataset
df = pd.read_csv('train.csv',encoding='latin1')


# Define stopwords and punctuation
stopwords_set = set(stopwords.words('english'))
punctuation = set(string.punctuation)


# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()


# Define a function for preprocessing text
def preprocess_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Tokenize the text
    tokens = word_tokenize(text.lower())

    # Remove stopwords and punctuation
    tokens = [token for token in tokens if token not in stopwords_set and token not in punctuation]

    # Lemmatize the tokens
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join the tokens back into a single string
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text




# Convert the 'selected_text' column to strings
df['selected_text'] = df['selected_text'].astype(str)

# Apply the preprocessing function to the 'selected_text' column
df['preprocessed_text'] = df['selected_text'].apply(preprocess_text)

# Print the preprocessed text
print(df['preprocessed_text'])
```

```
0                         responded going
1                                sooo sad
2                                bullying
3                             leave alone
4                                     son
                        ...
27476                              lost
27477                             force
27478                            yay good
27479                               worth
27480    flirting going atg smile yay hug
Name: preprocessed_text, Length: 27481, dtype: object
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import VotingClassifier
from sklearn.pipeline import Pipeline
import numpy as np



# Split the dataset into training and test sets
X = df['preprocessed_text']
y = df['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)




# Create a pipeline for vectorization, model selection, and hyperparameter tuning
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer()),
    ('classifier', RandomForestClassifier())
])

parameters = {
    'vectorizer__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'vectorizer__min_df': [1, 2, 3],
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [None, 5, 10],
}



random_search = RandomizedSearchCV(pipeline, parameters, cv=5, n_iter=3, random_state=42)
random_search.fit(X_train, y_train)
```




```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Encode the target variables
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)



oversampler = SMOTE(random_state=42)



# Fit and transform the training text data
X_train_vectorized = vectorizer.fit_transform(X_train)

# Apply SMOTE to the vectorized training data
X_train_oversampled, y_train_oversampled = oversampler.fit_resample(X_train_vectorized, y_train_encoded)



X = df['preprocessed_text']



from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer()

# Fit and transform the training text data
X_train_vectorized = vectorizer.fit_transform(X_train)

# Transform the test text data
X_test_vectorized = vectorizer.transform(X_test)
```

```
# Apply SMOTE to the vectorized training data
X_train_oversampled, y_train_oversampled = oversampler.fit_resample(X_train_vectorized, y_train_encoded)

# Fit the ensemble model on the oversampled training data
ensemble.fit(X_train_oversampled, y_train_oversampled)

# Predict the target variable for the test set
y_pred_ensemble = ensemble.predict(X_test_vectorized)



# Create an ensemble of models
model1 = RandomForestClassifier()
model2 = MultinomialNB()
ensemble = VotingClassifier(estimators=[('rf', model1), ('nb', model2)], voting='hard')



# Print the best parameters and the classification report
print("Best Parameters: ", random_search.best_params_)
y_pred = random_search.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Best Parameters:  {'vectorizer__ngram_range': (1, 1), 'vectorizer__min_df': 1, 'classifier__n_estimators': 100, 'classifier__max_de
              precision    recall  f1-score   support

    negative       0.76      0.75      0.76      1562
     neutral       0.79      0.84      0.81      2230
    positive       0.87      0.79      0.82      1705

    accuracy                           0.80      5497
   macro avg       0.80      0.79      0.80      5497
weighted avg       0.80      0.80      0.80      5497
```

0s completed at 1:16 AM