AI assignment – NLP
Sentiment analysis

1) Use the twitter data from the **nltk** library in python

Code :

```
[37] import nltk
     from nltk.corpus import twitter_samples

     # Download NLTK resources
     nltk.download('twitter_samples')
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

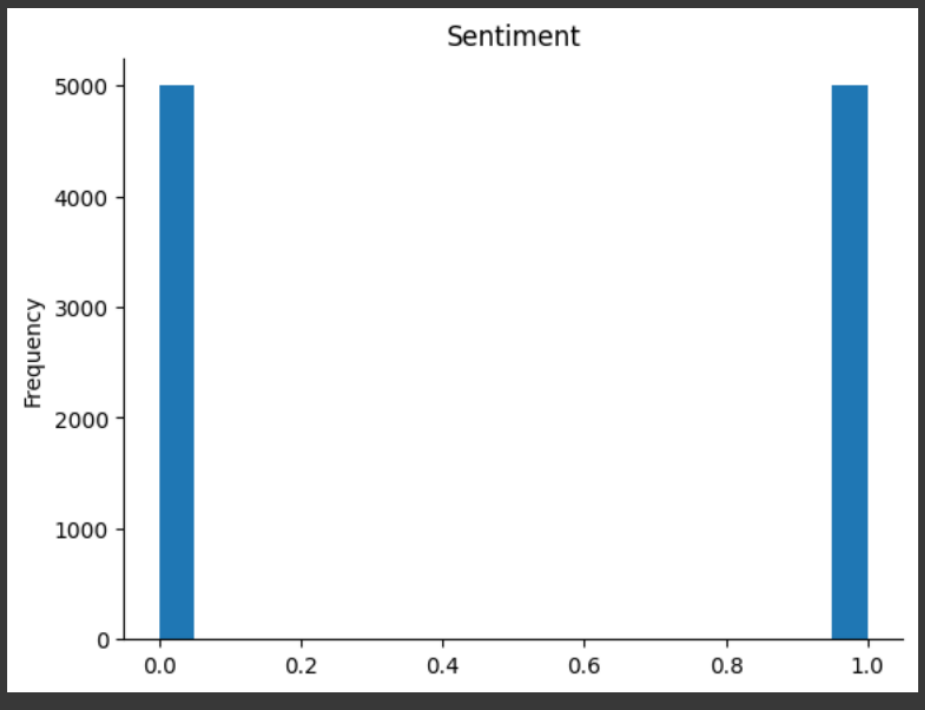
Preparing the data

```
[ ] from nltk.corpus import twitter_samples
    positive_tweets = twitter_samples.strings('positive_tweets.json')
```

```
[ ] negative_tweets = twitter_samples.strings('negative_tweets.json')
```

```
import pandas as pd
# creating a dataframe for positive tweets and adding a sentiment column with ones
df = pd.DataFrame(positive_tweets, columns=['Tweet'])
df['Sentiment'] = 1

# creating a dataframe for negative tweets and adding a sentiment column with zeroes
temp_df = pd.DataFrame(negative_tweets, columns=['Tweet'])
temp_df['Sentiment'] = 0

# appending the negative dataframe to positive, index is reset after appending
df = pd.concat([df, temp_df], ignore_index=True)

# shuffles the rows of the dataframe randomly
df = df.sample(frac = 1)

# resets the index after shuffling
df.reset_index(drop=True, inplace=True)
# df
```

```python
from matplotlib import pyplot as plt
df['Sentiment'].plot(kind='hist', bins=20, title='Sentiment')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Cleaning the data

```python
# Converting every sentence to lowercase
df['Tweet'] = df['Tweet'].apply(lambda x: x.lower())

# removing urls
import re
def remove_urls(Tweet):
 Tweet = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\'(?:%[0-9a-fA-F][0-9a-fA-F]))+','', Tweet)
 return Tweet

df['Tweet'] = df['Tweet'].apply(lambda x: remove_urls(x))


# Removing twitter handles, punctuation, extra spaces, numbers and special characters
import string
import re

def remove_noise(tweet):
    tweet = re.sub(r"(@[A-Za-z0-9_]+)", "", tweet)
    tweet = "".join([char if char not in string.punctuation else " " for char in tweet])
    tweet = re.sub(r' +', ' ', tweet)
    tweet = re.sub(r"[0-9]+", "", tweet)
    tweet = re.sub(r"[^A-Za-z0-9_. ]+", "", tweet)
    return tweet

df['Tweet'] = df['Tweet'].apply(lambda x: remove_noise(x))
```

```
# removing stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

  # Loading stop words and removing negative stop words from the list
stop_words = stopwords.words('english')
words_to_keep = ['don', "don't", 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
                 "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
                 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'no',
                 'nor', 'not', 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
                 "won't", 'wouldn', "wouldn't"]
my_stop_words = stop_words
for word in words_to_keep:
  my_stop_words.remove(word)

  # Removing stop words from the Tweet
def remove_stop_words(Tweet):
 tokens = word_tokenize(Tweet)
 Tweet_with_no_stop_words = [token for token in tokens if not token in my_stop_words]
 reformed_Tweet =' '.join(Tweet_with_no_stop_words)
 return reformed_Tweet

df['Tweet'] = df['Tweet'].apply(lambda x: remove_stop_words(x))

# stemming
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

def lemmatise_sentence(Tweet):
 token_words = word_tokenize(Tweet)
 lemmatized_Tweet = []
 for word in token_words:
  lemmatized_Tweet.append(stemmer.stem(word))
  lemmatized_Tweet.append(" ")
 return "".join(lemmatized_Tweet)
df['Tweet'] = df['Tweet'].apply(lambda x: lemmatise_sentence(x))
```

training and testing

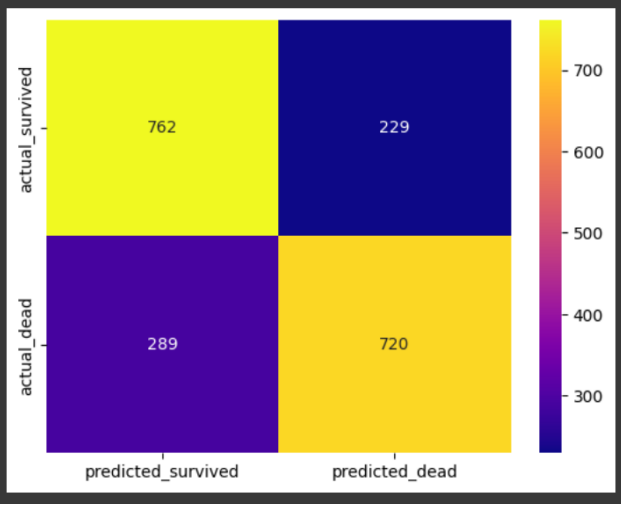
```
[43] import pandas as pd
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     # Split dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(df['Tweet'], df['Sentiment'], test_size=0.2, random_state=42)
     # Vectorize tweets
     vectorizer = CountVectorizer()
     X_train = vectorizer.fit_transform(X_train)
     X_test = vectorizer.transform(X_test)
```

```
[44] # Logistic Regression Classifier
     lr = LogisticRegression()
     lr.fit(X_train, y_train)
     y_pred = lr.predict(X_test)
     logistic_accuracy = lr.score(X_test, y_test)
     print(logistic_accuracy)
```

```
0.7425
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['actual_survived', 'actual_dead'], yticklabels=['predicted_survived', 'predicted_dea
            , annot=True, fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
print('\n')
```

```
Accuracy: 0.741
Precision: [0.72502379 0.75869336]
Recall: [0.76892028 0.7135778 ]
F1 Score: [0.74632713 0.73544433]
```

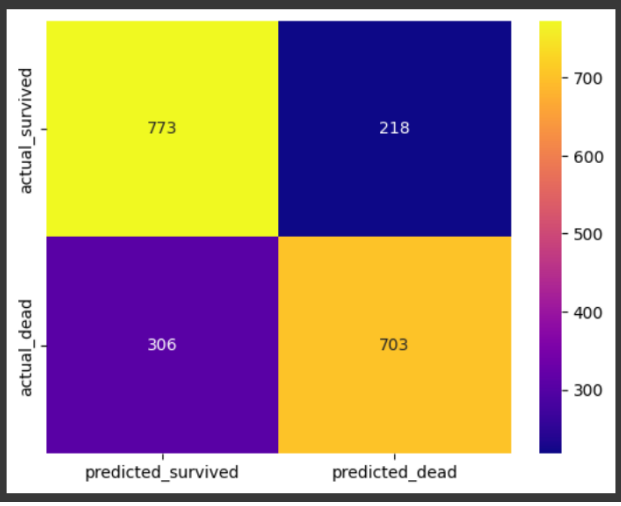



```python
# Naive Bayes Classifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
nb_accuracy = nb.score(X_test,y_test)
print(nb_accuracy)
```

```
0.7465
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['actual_survived', 'actual_dead'], yticklabels=['predicted_survived', 'predicted_dead']
            , annot=True, fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
print('\n')
```

```
Accuracy: 0.738
Precision: [0.71640408 0.76330076]
Recall: [0.78002018 0.69672944]
F1 Score: [0.7468599  0.72849741]
```

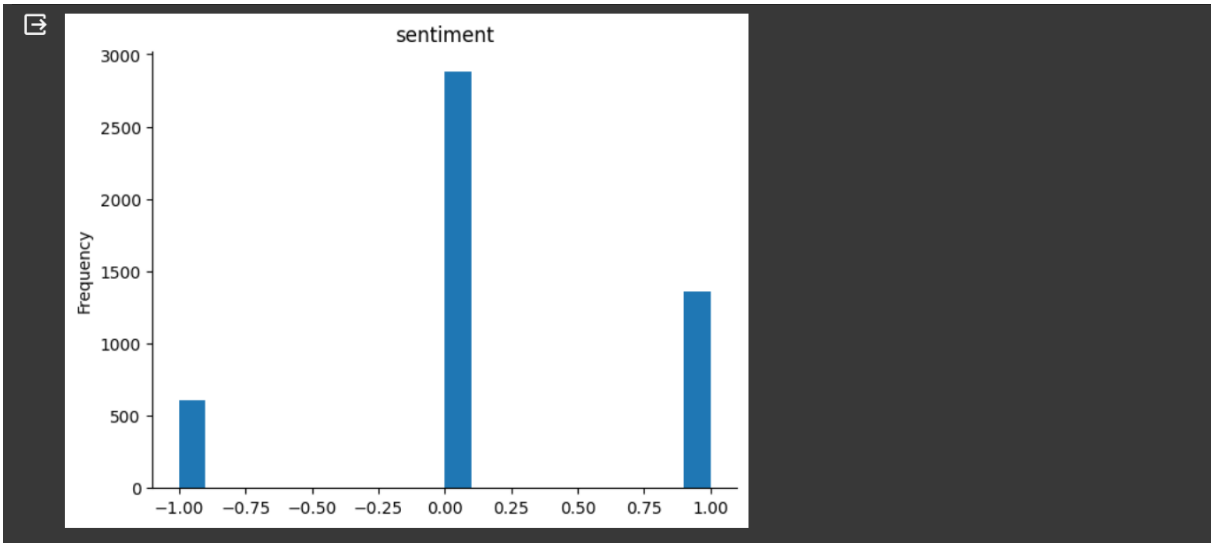


2) Financial data (financial.csv)

Code :

```
[ ]  import pandas as pd
     financial = pd.read_csv('./financial.csv',encoding='ISO-8859-1',names=['sentiment','text'])
     financial.dropna(inplace=True)
     financial['sentiment'].replace({'neutral':0,'positive':1,'negative':-1},inplace=True)
     # financial
```

```
     from matplotlib import pyplot as plt
     financial['sentiment'].plot(kind='hist', bins=20, title='sentiment')
     plt.gca().spines[['top', 'right',]].set_visible(False)
```



Cleaning the data

```
# Converting every sentence to lowercase
financial['text'] = financial['text'].apply(lambda x: x.lower())

# removing special charecters
import re

def remove_special_characters(text):
    pattern = r'[^a-zA-Z0-9\s]'
    text = re.sub(pattern, '', text)
    return text

financial['text'] = financial['text'].apply(lambda x: remove_special_characters(x))

# removing stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

  # Loading stop words and removing negative stop words from the list
stop_words = stopwords.words('english')
words_to_keep = ['don', "don't", 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't"
my_stop_words = stop_words
for word in words_to_keep:
  my_stop_words.remove(word)

  # Removing stop words from the text
def remove_stop_words(text):
 tokens = word_tokenize(text)
 text_with_no_stop_words = [token for token in tokens if not token in my_stop_words]
 reformed_text =' '.join(text_with_no_stop_words)
 return reformed_text

financial['text'] = financial['text'].apply(lambda x: remove_stop_words(x))

# stemming
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

def lemmatise_sentence(text):
 token_words = word_tokenize(text)
 lemmatized_text = []
 for word in token_words:
  lemmatized_text.append(stemmer.stem(word))
  lemmatized_text.append(" ")
 return "".join(lemmatized_text)
financial['text'] = financial['text'].apply(lambda x: lemmatise_sentence(x))
# print(financial)
```

Training and Testing

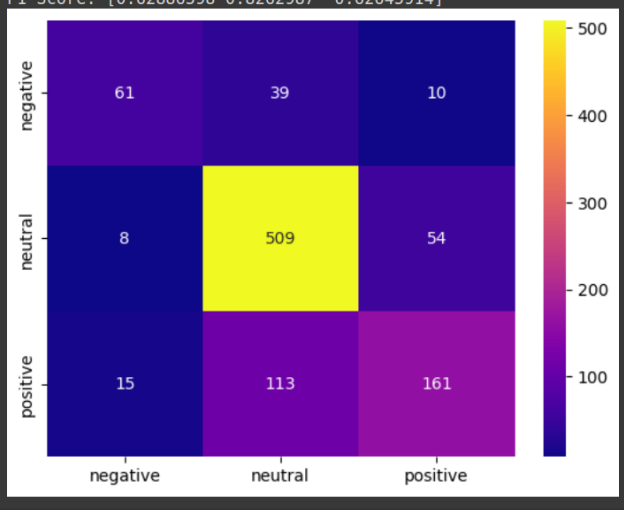
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(financial['text'], financial['sentiment'],
                                                    test_size=0.2, random_state=42)
# Vectorize text
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

```
[ ] # Logistic Regression Classifier
    lr = LogisticRegression(max_iter = 1000)
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    logistic_accuracy = lr.score(X_test, y_test)
    print(logistic_accuracy)
```

```
0.7536082474226804
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['negative','neutral','positive'], yticklabels=['negative','neutral','positive'],
            annot=True, fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
```

```
Accuracy: 0.754
Precision: [0.72619048 0.77004539 0.71555556]
Recall: [0.55454545 0.89141856 0.55709343]
F1 Score: [0.62886598 0.8262987  0.62645914]
```

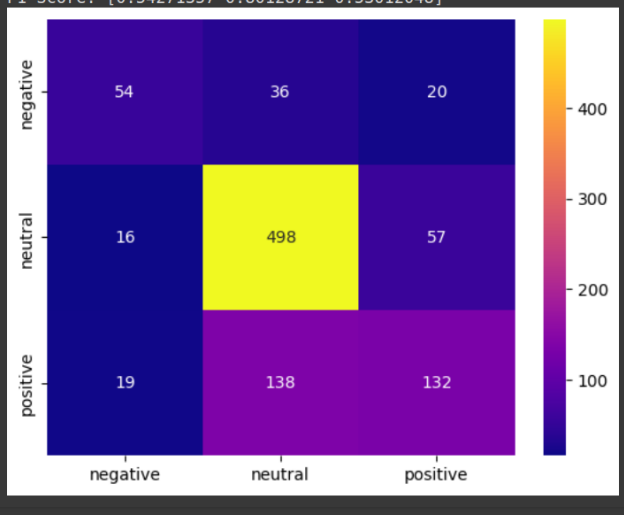


```
[ ]  from sklearn.naive_bayes import MultinomialNB

     nb = MultinomialNB()
     nb.fit(X_train, y_train)
     y_pred = nb.predict(X_test)
     nb_accuracy = nb.score(X_test,y_test)
     print(nb_accuracy)
```

```
0.7051546391752578
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['negative','neutral','positive'], yticklabels=['negative','neutral','positive'],
            annot=True, fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.705
Precision: [0.60674157 0.74107143 0.63157895]
Recall: [0.49090909 0.87215412 0.4567474 ]
F1 Score: [0.54271357 0.80128721 0.53012048]
```

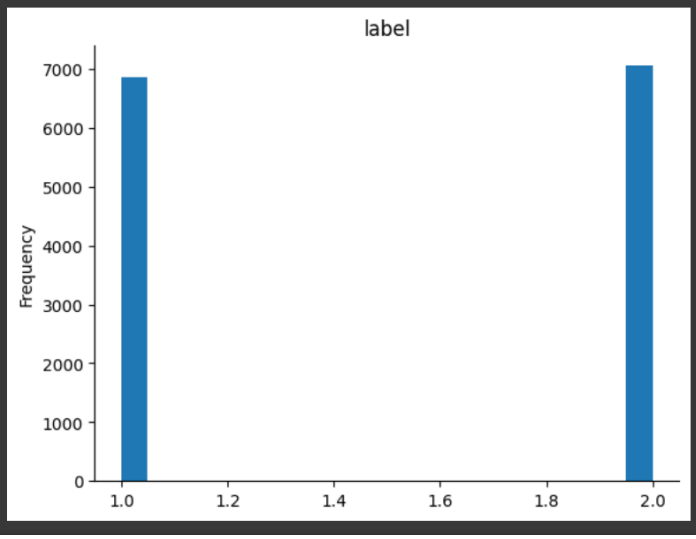


3) Product review dataset

Code :

```
[ ]  import pandas as pd

     # Read the text file
     with open('dataset.txt', 'r') as file:
         data = file.readlines()

     # Split each line into text and sentiment label
     texts = []
     labels = []
     for line in data:
         label, text = line.strip().split(' ', 1)
         texts.append(text)
         labels.append(int(label.strip('__label__')))

     product = pd.DataFrame({'text':texts, 'label':labels})
     # product
```

```
from matplotlib import pyplot as plt
product['label'].plot(kind='hist', bins=20, title='label')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Cleaning the data / preprocessing

```
# Converting every sentence to lowercase
product['text'] = product['text'].apply(lambda x: x.lower())

# removing special charecters
import re

def remove_special_characters(text):
    pattern = r'[^a-zA-Z0-9\s]'
    text = re.sub(pattern, '', text)
    return text

product['text'] = product['text'].apply(lambda x: remove_special_characters(x))

# removing stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = stopwords.words('english')
def remove_stop_words(text):
 tokens = word_tokenize(text)
 text_with_no_stop_words = [token for token in tokens if not token in stop_words]
 reformed_text =' '.join(text_with_no_stop_words)
 return reformed_text

product['text'] = product['text'].apply(lambda x: remove_stop_words(x))
```

```
# stemming
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

def lemmatise_sentence(text):
 token_words = word_tokenize(text)
 lemmatized_text = []
 for word in token_words:
  lemmatized_text.append(stemmer.stem(word))
  lemmatized_text.append(" ")
 return "".join(lemmatized_text)
product['text'] = product['text'].apply(lambda x: lemmatise_sentence(x))
# product
```

Training and Testing

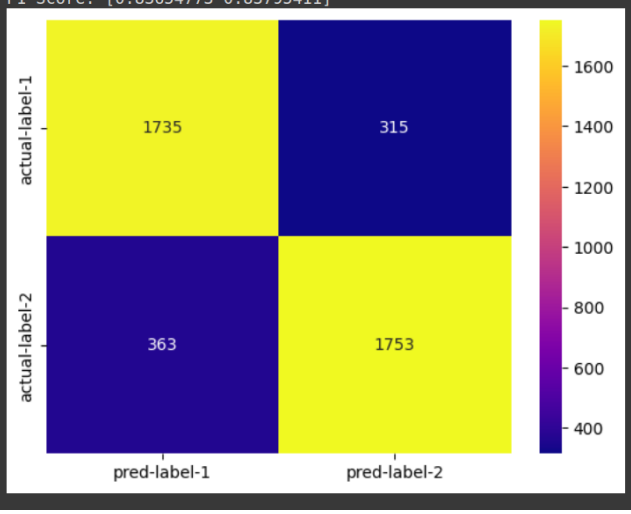
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(product['text'], product['label'], test_size=0.2, random_state=

# Vectorize text
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
nb_accuracy = nb.score(X_test,y_test)
print(nb_accuracy)
```

```
0.8224937118217751
```

Here I did naive bias classification.

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['actual-label-1','actual-label-2'],
            yticklabels=['pred-label-1','pred-label-2'], annot=True, fmt='d',
            annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.837
Precision: [0.82697807 0.84767892]
Recall: [0.84634146 0.82844991]
F1 Score: [0.83654773 0.83795411]
```

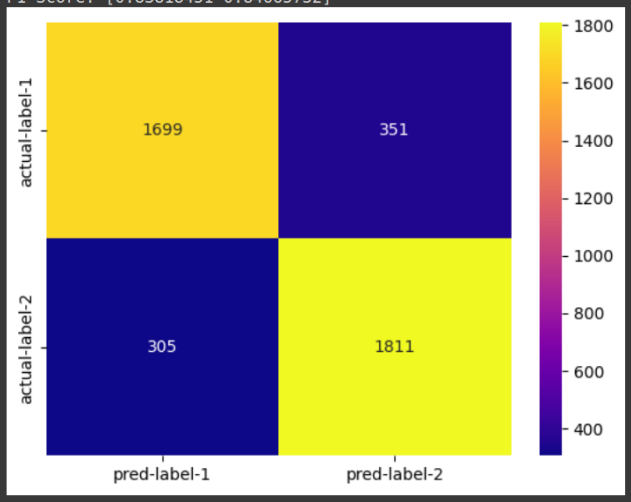


```python
from sklearn.linear_model import LogisticRegression

# Logistic Regression Classifier

lr = LogisticRegression(max_iter = 1000)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
logistic_accuracy = lr.score(X_test, y_test)
print(logistic_accuracy)
```

```
0.8393819619116062
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['actual-label-1','actual-label-2'],
            yticklabels=['pred-label-1','pred-label-2'], annot=True,
            fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.843
Precision: [0.84780439 0.83765032]
Recall: [0.82878049 0.85586011]
F1 Score: [0.83818451 0.84665732]
```



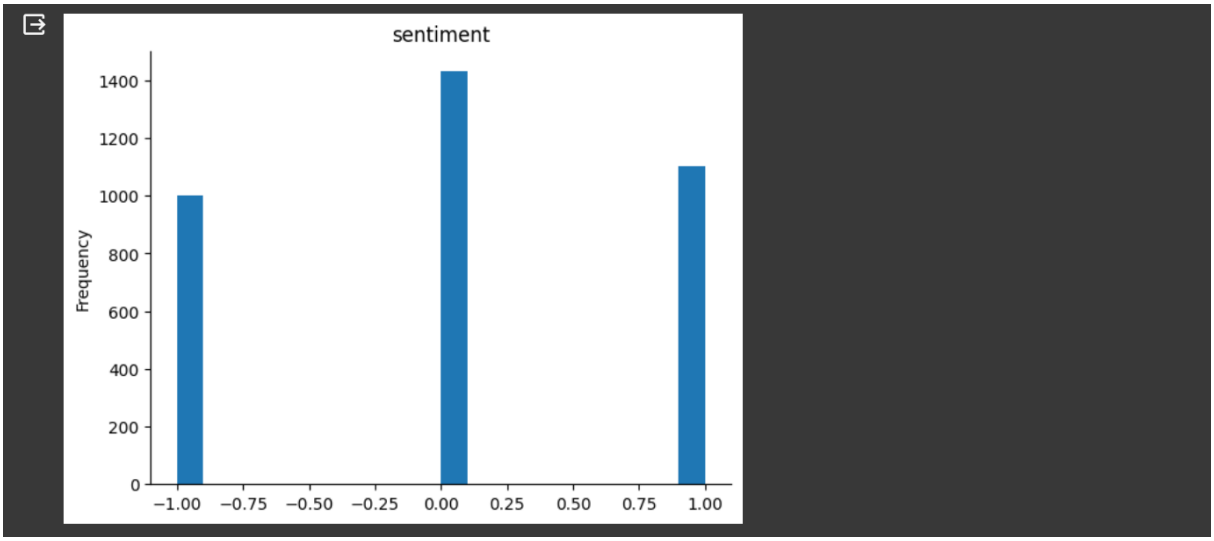
4) Emotions dataset

test data

```
[ ] import pandas as pd
    test = pd.read_csv('./test.csv',encoding='ISO-8859-1')
    train = pd.read_csv('./train.csv',encoding='ISO-8859-1')
```

```
test.drop(['textID','Age of User','Time of Tweet','Population -2020','Country','Density (P/Km²)','Land Area (Km²)'],
          axis=1,inplace=True)
test.dropna(inplace=True)
test['sentiment'].replace({'neutral':0,'positive':1,'negative':-1},inplace=True)
# test
```

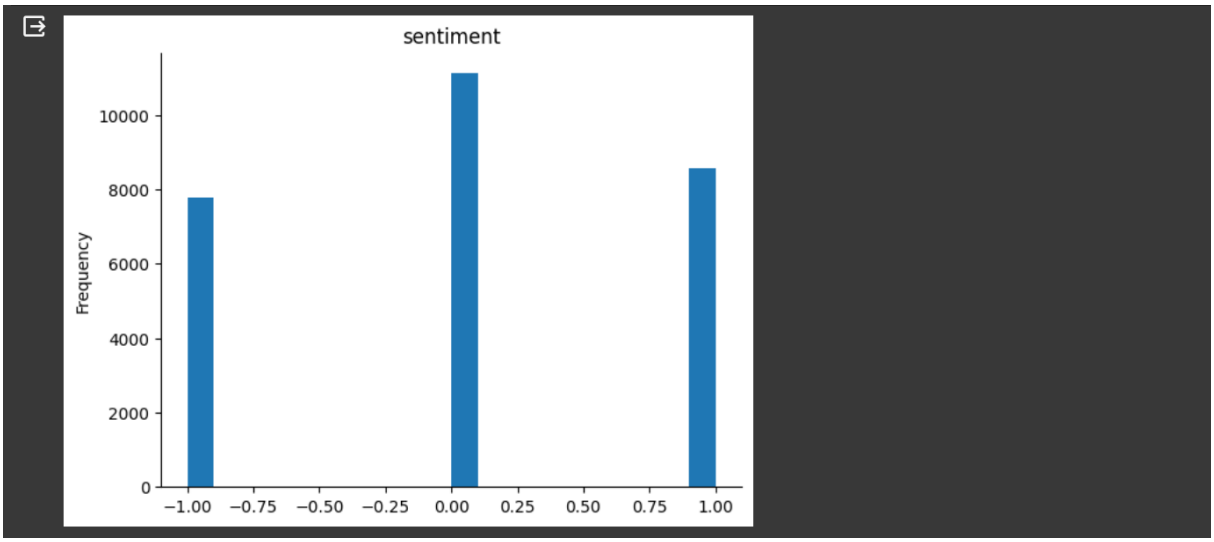
```
[ ] from matplotlib import pyplot as plt
    test['sentiment'].plot(kind='hist', bins=20, title='sentiment')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```



train data

```
train.drop(['Unnamed: 0','Age of User','Time of Tweet','Population -2020','Country','Density (P/Km²)',
            'Land Area (Km²)','selected_text'],axis=1,inplace=True)
train.dropna(inplace=True)
train['sentiment'].replace({'neutral':0,'positive':1,'negative':-1},inplace=True)
# train
```

```
[ ] from matplotlib import pyplot as plt
    train['sentiment'].plot(kind='hist', bins=20, title='sentiment')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```



Cleaning the text data

```
# Converting every sentence to lowercase
train['text'] = train['text'].apply(lambda x: x.lower())
test['text'] = test['text'].apply(lambda x: x.lower())

# removing urls
import re
def remove_urls(text):
 text = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\'(?:%[0-9a-fA-F][0-9a-fA-F]))+','', text)
 return text

train['text'] = train['text'].apply(lambda x: remove_urls(x))
test['text'] = test['text'].apply(lambda x: remove_urls(x))


# Removing twitter handles, punctuation, extra spaces, numbers and special characters
import string
import re

def remove_noise(tweet):
    tweet = re.sub(r"(@[A-Za-z0-9_]+)", "", tweet)
    tweet = "".join([char if char not in string.punctuation else " " for char in tweet])
    tweet = re.sub(r' +', ' ', tweet)
    tweet = re.sub(r"[0-9]+", "", tweet)
    tweet = re.sub(r"[^A-Za-z0-9_. ]+", "", tweet)
    return tweet

train['text'] = train['text'].apply(lambda x: remove_noise(x))
test['text'] = test['text'].apply(lambda x: remove_noise(x))
```

```
# removing stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

  # Loading stop words and removing negative stop words from the list
stop_words = stopwords.words('english')
words_to_keep = ['don', "don't", 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
                 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't",
                 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn',
                "needn't", 'shan', "shan't", 'no', 'nor', 'not', 'shouldn', "shouldn't",
                 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
my_stop_words = stop_words
for word in words_to_keep:
  my_stop_words.remove(word)

  # Removing stop words from the text
def remove_stop_words(text):
 tokens = word_tokenize(text)
 text_with_no_stop_words = [token for token in tokens if not token in my_stop_words]
 reformed_text =' '.join(text_with_no_stop_words)
 return reformed_text

train['text'] = train['text'].apply(lambda x: remove_stop_words(x))
test['text'] = test['text'].apply(lambda x: remove_stop_words(x))
```

```
# stemming
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

def lemmatise_sentence(text):
 token_words = word_tokenize(text)
 lemmatized_text = []
 for word in token_words:
  lemmatized_text.append(stemmer.stem(word))
  lemmatized_text.append(" ")
 return "".join(lemmatized_text)
train['text'] = train['text'].apply(lambda x: lemmatise_sentence(x))
test['text'] = test['text'].apply(lambda x: lemmatise_sentence(x))
```

Training and Testing

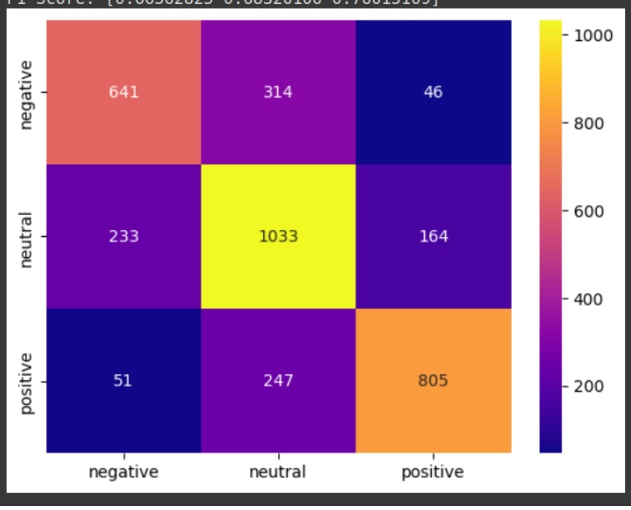
```
[ ] import pandas as pd
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.linear_model import LogisticRegression

    vectorizer = CountVectorizer()
    X_train = vectorizer.fit_transform(train['text'])
    y_train = train['sentiment']
    X_test = vectorizer.transform(test['text'])
    y_test = test['sentiment']

    lr = LogisticRegression(max_iter=1731)
    lr.fit(X_train,y_train)
    y_pred = lr.predict(X_test)
    logistic_accuracy = lr.score(X_test, y_test)
    print(logistic_accuracy)
```

```
0.7014714204867006
```

Here I did logistic regression

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['negative','neutral','positive'],
            yticklabels=['negative','neutral','positive'], annot=True,
            fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.701
Precision: [0.69297297 0.64805521 0.79310345]
Recall: [0.64035964 0.72237762 0.72982774]
F1 Score: [0.66562825 0.68320106 0.76015109]
```



```
[ ] from sklearn.naive_bayes import MultinomialNB

    nb = MultinomialNB()
    nb.fit(X_train, y_train)
    y_pred = nb.predict(X_test)
    nb_accuracy = nb.score(X_test,y_test)
    print(nb_accuracy)
```

```
0.6615732880588568
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

cm = confusion_matrix(y_test,y_pred)

accuracy = np.trace(cm) / np.sum(cm)
precision = np.diag(cm) / np.sum(cm, axis=0)
recall = np.diag(cm) / np.sum(cm, axis=1)
f1 = 2 * (precision * recall) / (precision + recall)

sns.heatmap(cm, xticklabels=['negative','neutral','positive'],
            yticklabels=['negative','neutral','positive'], annot=True,
            fmt='d', annot_kws={'fontsize': 10}, cmap="plasma")

print('Accuracy: {:.3f}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print('F1 Score: {}'.format(f1))
```

```
Accuracy: 0.662
Precision: [0.68638393 0.6025958  0.73333333]
Recall: [0.61438561 0.68181818 0.6781505 ]
F1 Score: [0.6483922  0.63976378 0.70466321]
```