In [ ]:

```
### Tokenize

review =  data.review_text
data_list = list()
for comp in review:
    data_list.append(RegexpTokenizer('\w+').tokenize(comp))

print(data_list)
```

In [ ]:

```
#### converting to lower case

text = " Lets Start learning NLP"
# python inbuild function
text = text.lower()
text
```

In [ ]:

```
#### Spelling Correction

from spellchecker import SpellChecker

def correct_spellings(text):
    spell = SpellChecker()
    corrected_words = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            corrected_words.append(spell.correction(word))
        else:
            corrected_words.append(word)
    return " ".join(corrected_words)

text = "this is speling correct tst"
print (correct_spellings(text))
```

In [ ]:

```
#### Stemming

from nltk.stem import LancasterStemmer
```

```
# LancasterStemmer
lancaster=LancasterStemmer()
stemmed=[]

for line in filtered_sentence:
    stemmed.append([lancaster.stem(x) for x in line])

print(len(filtered_sentence[9]))
print(len(stemmed[9]))

# PorterStemmer
import nltk
def simple_stemmer(text):
    ps = nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text

simple_stemmer("My system keeps crashing his crashed yesterday, ours crashes daily")
```

In [ ]:

```
#### Lemitization

wordNetLemmatizer = WordNetLemmatizer()
lemmitized=[]

for line in stemmed:
    lemmitized.append([wordNetLemmatizer.lemmatize(word) for word in line])

print(len(stemmed[0]))
print(len(lemmitized[0]))
```

In [ ]:

```
#### Removing multiple spaces

#Converting line with mutiple Spaces into line with single space b/w words
import re
text = "Converting line   with    many   spaces to     line with single space between words."
text = re.sub(' +',' ',text)
text
```

In [ ]:

```
## Removng digits
```

```
text = "Being no 1 in exam is more important or being no 3   with fair ways "
text = re.sub(r'[0-9]','',text)
print(text)
```

In [ ]:

```
### Removing stop words

from nltk.corpus import stopwords
#nltk.download('stopwords')
text = ["Stoword is one if the important topic"]

stop_words = set(stopwords.words('english'))
filtered_sentence = []

for lines in text:
    word = [w for w in lines if w not in stop_words]
    filtered_sentence.append(word)

print(len(text[0]))
print(len(filtered_sentence[0]))
```

In [ ]:

```
### Removing punctuations

import string
string.punctuation

text = "This! sentence, contains so: many - punctuations."
text = text.translate(str.maketrans('', '', string.punctuation))
print(text)
```

In [ ]:

```
### Remove URLs

text = 'Shall I search the answer in www.google.com ?'
text  = re.sub(r"https?://\S+|www\.\S+", "", text )
print(text)
```

In [ ]:

```
### Remove accented text

import unicodedata
def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text
remove_accented_chars('Sómě Áccěntěd těxt')
```

In [ ]:

```
### Word cloud

#! pip install wordcloud
import nltk
#nltk.download('stopwords')

from nltk.corpus import stopwords

stopwords = set(stopwords.words('english'))

# import the wordcloud library
from wordcloud import WordCloud,STOPWORDS

# Instantiate a new wordcloud.
wordcloud = WordCloud(random_state = 8,
        normalize_plurals = False,
        width = 600, height= 300,
        max_words = 300,
#          background_color='white',
        stopwords = stopwords)

# Apply the wordcloud to the text.
text = '''Barack Obama is an American politician who served as the 44th President of the United States
from 2009 to 2017.He is the first African American to have served as president,
as well as the first born outside the contiguous United States. He speaks English.'''

wordcloud.generate(text)

# ploting wordcloud
import matplotlib.pyplot as plt

# create a figure
fig, ax = plt.subplots(1,1, figsize = (9,6))

# add interpolation = bilinear to smooth things out
plt.imshow(wordcloud, interpolation='bilinear')
```

```
# and remove the axis
plt.axis("off")
```

In [ ]:

```
### Sentiment analysis with TextBlob

from textblob import TextBlob

text = "I hate anything that goes in my ear"
textblob = TextBlob(text)

# fetching text sentiment polarity
textblob.sentiment.polarity

# fetching text sentiment subjectivity
textblob.sentiment.subjectivity
```

In [ ]:

```
# Assign the subjectivity response to the content
from textblob import TextBlob

review_df['Subjectivity'] = review_df['Review'].apply(lambda x: TextBlob((str(x))).sentiment.subjectivity)
review_df['Subjectivity'].head(2)
```

In [ ]:

```
# Assign the polarity response to the content
def getTextPolarity(txt):
    return TextBlob(txt).sentiment.polarity

review_df['Polarity'] = review_df['Review'].transform(lambda x: getTextPolarity(str(x)))
review_df['Polarity'].head(2)
```

In [ ]:

```
# Assing sentiment to the content using polity
# same can be done using subjetivity its reange would be [0,1]
def getTextAnalysis(a):
    if a < 0:
        return "Negative"
    elif a == 0:
        return "Neutral"
    else:
        return "Positive"
```

```
review_df['Sentiment'] = review_df['Polarity'].apply(getTextAnalysis)

positive = review_df[review_df['Sentiment'] == 'Positive']

print(str(positive.shape[0]/(review_df.shape[0])*100) + " % of positive Review")
```

In [ ]:

```
# Visualize the frequency distribution of the sentiment on each content

plt.figure(figsize = (10,5))
labels = review_df.groupby('Sentiment').count().index.values

values = review_df.groupby('Sentiment').size().values

plt.bar(labels, values)
```

TF-IDF

In [ ]:

```
# convert list of words into sentence, lematized text is an array

def list_to_text(list):
    return ' '.join([str(elem) for elem in list])

review_df['Review_sentences'] = review_df['Review'].apply(list_to_text)
review_df.head(2)
```

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

corpus = review_df['Review_sentences']

vectorizer.fit(corpus)

skl_tf_idf_vectorized = vectorizer.transform(corpus)
print(skl_tf_idf_vectorized.toarray())
```

In [ ]:

```python
skl_tfdf_output = (skl_tf_idf_vectorized.toarray())[0] # 0: As we are interested only in the first record
print(skl_tfdf_output)
```

In [ ]:

```python
df_tfdf_sklearn = pd.DataFrame(skl_tfdf_output, index = vectorizer.get_feature_names_out(), columns=['tf-idf'])

df_tfdf_sklearn.loc['<any keyword from the document>']
```

Applying a Machine Learning (ML) algorithm to the output of a TfidfVectorizer involves the following steps:

Prepare the data: Preprocess the text data and split it into training and testing sets.

Transform text data using TfidfVectorizer: Convert the text into numeric features.

Train the ML model: Use the TF-IDF matrix as input for a supervised or unsupervised ML model.

Evaluate the model: Assess the performance of the model using suitable metrics.

Example: Applying a Classifier to TF-IDF Features

In [ ]:

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:

```python
# Sample dataset
data = {
    'text': [
        'I love this product. It works great!',
        'This is the worst product I ever bought.',
        'Amazing quality and fantastic customer service.',
        'Terrible experience. I want a refund.',
        'Good value for the price.',
        'Poor quality and not worth the money.'
    ],
    'label': [1, 0, 1, 0, 1, 0]  # 1: Positive, 0: Negative
}

df = pd.DataFrame(data)

# Split data into features and labels
X = df['text']
y = df['label']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Step 3: Transform Text Data Using TfidfVectorizer

In [ ]:

```
# Create and fit the TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Transform the training and test sets
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [ ]:

```
X_train_tfidf.toarray()
```

Step 4: Train an ML Model

In [ ]:

```
# Initialize and train a Random Forest Classifier
clf = XGBClassifier(random_state=7)
clf.fit(X_train_tfidf, y_train)
```

Step 5: Make Predictions and Evaluate the Model

In [ ]:

```
# Make predictions on the test data
y_pred = clf.predict(X_test_tfidf)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

LSTM

In [26]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [28]:

```
# Example text data
texts = [
    "This movie is great!",
    "I did not like the acting.",
    "The plot was thrilling and engaging.",
    "The characters felt flat and uninspired."
]

# Labels for sentiment (1 = positive, 0 = negative)
labels = [1, 0, 1, 0]
```

In [30]:

```
# Tokenize words
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to ensure uniform length
maxlen = 100  # Example maximum sequence length
X = pad_sequences(sequences, maxlen=maxlen)

# Convert labels to categorical
y = tf.keras.utils.to_categorical(labels)
```

In [32]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
```

Out[32]:

```
<keras.src.legacy.preprocessing.text.Tokenizer at 0x154e26240>
```

In [ ]:

```python
# Define the LSTM model
embedding_dim = 100
model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=embedding_dim, input_length=maxlen))
model.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(units=2, activation='softmax'))  # Binary classification

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Print the model summary
model.summary()
```

In [ ]:

```python
# Train the model
batch_size = 32
epochs = 10

# Using a small dataset, use a higher validation split (e.g., 50%)
model.fit(X, y, batch_size=batch_size, epochs=epochs, validation_split=0.5)
```

In [ ]:

```python
# Example: Evaluating the model with test data
# Replace these with your actual test texts and labels
test_texts = [
    "I loved the storyline!",
    "The acting was terrible."
]
test_labels = [1, 0]

# Prepare test data
test_sequences = tokenizer.texts_to_sequences(test_texts)
X_test = pad_sequences(test_sequences, maxlen=maxlen)
y_test = tf.keras.utils.to_categorical(test_labels)

# Evaluate the model
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```