In [16]:

```
import nltk
from nltk.tokenize import sent_tokenize
```

In [19]:

```
text= """India Is my Country """
```

In [21]:

```
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['India Is my Country']
```

In [22]:

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['India', 'Is', 'my', 'Country']
```

In [25]:

```
from nltk.corpus import stopwords
stopwords =set(stopwords.words("english"))
print(stopwords)
```

```
{'for', "that'll", 'an', 'is', 'up', 'some', 'off', 've', 'there', "wouldn't", 'won', 'into', 'and', 'are', 'your',
"he'd", 'hasn', 'our', 'than', 'under', 'doesn', 'any', 'o', "shouldn't", 'him', "i'll", 'my', 'of', 'over', "they'l
l", 'ourselves', "you're", 'his', 'which', 'own', "won't", 'down', 'each', 'been', 'll', 'when', 'all', 'both', "cou
ldn't", 's', "shan't", 'other', 'just', 'her', 'do', 'weren', 'herself', 'if', 'at', 'has', 'theirs', 'it', "you'd",
'aren', 'then', 'we', 'shouldn', 'only', 'themselves', 'this', 'again', 'to', 'as', "aren't", "it'd", 't', 'couldn',
'out', "doesn't", 'between', "hadn't", "we'd", 'be', 'were', 'does', 'those', "she'll", "isn't", 're', 'too', 'who',
'once', 'having', "mightn't", 'did', 'was', "he'll", 'had', 'whom', 'while', 'few', 'you', "they're", 'he', 'me', 's
uch', 'until', 'wasn', "needn't", 'she', "wasn't", 'below', 'further', 'its', 'that', 'can', "we're", 'shan', 'd',
"she'd", "it'll", 'against', 'no', 'they', 'very', 'same', 'mightn', "weren't", 'itself', 'yourself', 'but', 'ours',
"i'd", 'haven', 'hers', 'i', "didn't", 'didn', 'am', 'isn', "he's", "i've", "you've", 'being', 'the', "hasn't", "i
t's", 'nor', "she's", 'during', "mustn't", 'should', 'more', 'where', "haven't", "you'll", 'mustn', "we'll", 'your
s', 'above', 'what', 'hadn', 'here', 'a', 'doing', 'm', 'these', 'most', 'y', 'don', 'needn', 'yourselves', "they'v
e", 'because', 'after', "we've", 'through', 'about', "they'd", 'himself', 'by', "i'm", 'how', 'from', 'their', "do
n't", 'myself', 'now', 'or', "should've", 'have', 'why', 'not', 'in', 'ma', 'ain', 'so', 'them', 'with', 'wouldn',
'on', 'before', 'will'}
```

In [31]:

```
filtered_sent=[]
for w in tokenized_text:
 if w not in stopwords:
    filtered_sent.append(w)
```

```
print("Tokenized Sentence:",tokenized_text)
print("Filterd Sentence:",filtered_sent)
```

```
Tokenized Sentence: ['India Is my Country']
Filterd Sentence: ['India Is my Country']
```

In [33]:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
ps = PorterStemmer()
stemmed_words=[]
for w in filtered_sent:
    stemmed_words.append(ps.stem(w))
```

In [34]:

```
print("Filtered Sentence:",filtered_sent)
print("Stemmed Sentence:",stemmed_words)
```

```
Filtered Sentence: ['India Is my Country']
Stemmed Sentence: ['india is my countri']
```

In [35]:

```
from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()
from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()
word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))
```

```
Lemmatized Word: fly
Stemmed Word: fli
```

In [40]:

```
from nltk import pos_tag
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/anku/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[40]:

```
True
```

In [42]:

```
sent = " Birds are flying "
tokens=nltk.word_tokenize(sent)
```

```
print(tokens)
```

```
['Birds', 'are', 'flying']
```

In [44]:

```
nltk.download('averaged_perceptron_tagger_eng')
nltk.pos_tag(tokens)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /home/anku/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

Out[44]:

```
[('Birds', 'NNS'), ('are', 'VBP'), ('flying', 'VBG')]
```

In [65]:

```
documents = [
    "Natural language processing helps computers understand human language.",
    "Artificial intelligence and NLP are closely related fields.",
    "Deep learning improves language models.",
    "Python is great for data science.",
    "Machine learning is a part of AI.",
    "I love data analysis and visualization using Python."
]
```

In [81]:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

labels = [0, 0, 0, 1, 0, 1]
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
```

In [82]:

```
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=42)
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("Y_train:",y_train)
print("Y_test:",y_test)
```

```
X_train: (4, 27)
X_test: (2, 27)
Y_train: [1, 0, 0, 1]
Y_test: [0, 0]
```

In [83]:

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

Out[83]:

```
▼ MultinomialNB
MultinomialNB()
```

In [87]:

```
y_pred = model.predict(X_test)
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

MultinomialNB Accuracy: 1.0
```

In [91]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
print(X)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 31 stored elements and shape (6, 27)>
  Coords	Values
  (0, 18)	0.33923274157733224
  (0, 13)	0.5563514017701056
  (0, 20)	0.33923274157733224
  (0, 9)	0.33923274157733224
  (0, 4)	0.33923274157733224
  (0, 24)	0.33923274157733224
  (0, 10)	0.33923274157733224
```

```
(1, 2)        0.408248290463863
(1, 12)       0.408248290463863
(1, 19)       0.408248290463863
(1, 3)        0.408248290463863
(1, 22)       0.408248290463863
(1, 7)        0.408248290463863
(2, 13)       0.39339984891428303
(2, 6)        0.4797475439396706
(2, 14)       0.39339984891428303
(2, 11)       0.4797475439396706
(2, 17)       0.4797475439396706
(3, 21)       0.44836665359771705
(3, 8)        0.5467790631887662
(3, 5)        0.44836665359771705
(3, 23)       0.5467790631887662
(4, 14)       0.501613008756558
(4, 16)       0.6117125098631682
(4, 0)        0.6117125098631682
(5, 21)       0.3546939600231492
(5, 5)        0.3546939600231492
(5, 15)       0.43254606386088895
(5, 1)        0.43254606386088895
(5, 26)       0.43254606386088895
(5, 25)       0.43254606386088895
```

In [ ]:

In [ ]: