```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download required NLTK resources
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('stopwords')

# Sample text
text = "Natural Language Processing is a fascinating field of AI."

# Tokenize the text
tokens = word_tokenize(text)

# Get stop words
stop_words = set(stopwords.words('english'))

# Filter tokens to include only stop words
stop_word_tokens = [word for word in tokens if word.lower() in stop_words]

# Perform POS tagging
pos_tags = nltk.pos_tag(stop_word_tokens)

print("POS Tags of Stop Words:", pos_tags)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
POS Tags of Stop Words: [('is', 'VBZ'), ('a', 'DT'), ('of', 'IN')]
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = ["Natural language processing is amazing.",
             "Machine learning and NLP go hand in hand.",
             "TF-IDF helps find important words in a document."]

# Compute TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Display results
print("Feature Names:", vectorizer.get_feature_names_out())
print("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

```
Feature Names: ['amazing' 'and' 'document' 'find' 'go' 'hand' 'helps' 'idf' 'important'
 'in' 'is' 'language' 'learning' 'machine' 'natural' 'nlp' 'processing'
 'tf' 'words']
TF-IDF Matrix:
 [[0.4472136  0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.4472136  0.4472136
  0.         0.         0.4472136  0.         0.4472136  0.
  0.        ]
 [0.         0.32311233 0.         0.         0.32311233 0.64622465
  0.         0.         0.         0.24573525 0.         0.
  0.32311233 0.32311233 0.         0.32311233 0.         0.
  0.        ]
 [0.         0.         0.36325471 0.36325471 0.         0.
  0.36325471 0.36325471 0.36325471 0.27626457 0.         0.
  0.         0.         0.         0.         0.         0.36325471
  0.36325471]]
```

```
from nltk import ngrams
from collections import Counter, defaultdict

# Sample text
text = "Natural language processing is fun and challenging."

# Generate N-grams
```

```
N = 2  # Change to 3 for Trigrams, etc.
tokens = text.lower().split()
n_grams = list(ngrams(tokens, N))

# Calculate probabilities
model = defaultdict(lambda: 0)
counts = Counter(n_grams)

for ngram, count in counts.items():
    prefix = ngram[:-1]
    total_prefix_count = sum(c for ng, c in counts.items() if ng[:-1] == prefix)
    model[ngram] = count / total_prefix_count

print("N-gram Probabilities:", dict(model))
```

```
N-gram Probabilities: {('natural', 'language'): 1.0, ('language', 'processing'): 1.0, ('processing', 'is'): 1.0, ('is', 'fun'): 1.0, ('f
```

```
from gensim.models import Word2Vec

# Sample sentences
sentences = [["natural", "language", "processing", "is", "fun"],
             ["machine", "learning", "and", "nlp", "go", "together"]]

# Train Word2Vec model
model = Word2Vec(sentences, vector_size=50, window=3, min_count=1, sg=1)

# Get vector for a word
vector = model.wv["language"]
print("Word Vector for 'language':", vector)
```

```
Word Vector for 'language': [ 0.00805391  0.00869487  0.01991474 -0.00894748 -0.00277853 -0.01463464
 -0.01939566 -0.01816051 -0.00204551 -0.01300658  0.00969946 -0.01232805
  0.00503837  0.00147888 -0.00678431 -0.00195845  0.01995825  0.01829177
 -0.00892366  0.01816605 -0.01128353  0.01186184 -0.00619444  0.0068635
  0.00603445  0.01380092 -0.00474777  0.01755007  0.01517886 -0.01909529
 -0.01601642 -0.01527579  0.00584651 -0.00558944 -0.01385904 -0.01625653
  0.01661836  0.00398098 -0.01865603 -0.00958543  0.00627348 -0.00942641
  0.01056169 -0.00846688  0.00528359 -0.01609137  0.01241977  0.00963778
  0.00157439  0.0060269 ]
```