```
import re

text = ("Contact us at support@example.com or call +91-9876543210."
"Visit our website https://www.mywebsite.org for details."
"Follow us on Twitter @TechGuru and use the hashtag #AI2025."
"Meeting scheduled on 28/07/2025. Beware of badword1 and badword2.")

import re

email_pattern = r'\b[\w.-]+?@\w+?\.\w+?\b'
url_pattern = r'https?://\S+|www\.\S+'
date_pattern = r'\b(?:\d{1,2}[-/]\d{1,2}[-/]\d{2,4}|\d{4}[-/]\d{1,2}
[-/]\d{1,2})\b'
phone_pattern = r'\+?\d{1,3}[-\s]?\(?\d{2,4}\)?[-\s]?\d{3}[-\s]?\d{4}'
hashtag_pattern = r'#\w+'
mention_pattern = r'@\w+'

offensive_pattern = [word for word in["badword1" , "badword2",
"spamword"]]
emails = re.findall(email_pattern, text)
urls = re.findall(url_pattern, text)
dates = re.findall(date_pattern, text)
phones = re.findall(phone_pattern, text)
hashtags = re.findall(hashtag_pattern, text)
mentions = re.findall(mention_pattern, text)


print("Emails:", emails)
print("URLs:", urls)
print("Dates:", dates)
print("Phones:", phones)
print("Hashtags:", hashtags)
print("Mentions:", mentions)
print("Offensive Words:", offensive_pattern)

Emails: ['support@example.com']
URLs: ['https://www.mywebsite.org']
Dates: ['28/07/2025']
Phones: ['+91-9876543210']
Hashtags: ['#AI2025']
Mentions: ['@example', '@TechGuru']
Offensive Words: ['badword1', 'badword2']

import nltk
from nltk.util import ngrams
nltk.download('punkt')
text = "The future of AI is bright and full of opportunities."
tokens = nltk.word_tokenize(text)
unigrams = list(ngrams(tokens, 1))
bigrams = list(ngrams(tokens, 2))
trigrams = list(ngrams(tokens, 3))
```

```
print('Unigrams:', unigrams)
print('Bigrams:', bigrams)
print('Trigrams:', trigrams)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
Unigrams: [('The',), ('future',), ('of',), ('AI',), ('is',),
('bright',), ('and',), ('full',), ('of',), ('opportunities',), ('.',)]
Bigrams: [('The', 'future'), ('future', 'of'), ('of', 'AI'), ('AI',
'is'), ('is', 'bright'), ('bright', 'and'), ('and', 'full'), ('full',
'of'), ('of', 'opportunities'), ('opportunities', '.')]
Trigrams: [('The', 'future', 'of'), ('future', 'of', 'AI'), ('of',
'AI', 'is'), ('AI', 'is', 'bright'), ('is', 'bright', 'and'),
('bright', 'and', 'full'), ('and', 'full', 'of'), ('full', 'of',
'opportunities'), ('of', 'opportunities', '.')]
```

```
import nltk
nltk.download('edit_distance')
from nltk.metrics import edit_distance
word1 = "kitten"
word2 = "sitting"
distance = edit_distance(word1, word2)
print(f"Edit Distance between '{word1}' and '{word2}':", distance)
```

```
Edit Distance between 'kitten' and 'sitting': 3
```

```
[nltk_data] Error loading edit_distance: Package 'edit_distance' not
[nltk_data]     found in index
```

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "Apple Inc. is planning to open a new office in Mumbai by
2026."
doc = nlp(text)

for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Apple Inc. ORG
Mumbai GPE
2026 DATE
```

```
import nltk, spacy
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')

text = "This is an example sentence, showing the effect of stopword
removal."
```

```
tokens = nltk.word_tokenize(text)

nltk_stopwords = set(stopwords.words('english'))
nltk_filtered = [word for word in tokens if word.lower() not in
nltk_stopwords]

# SpaCy Stopwords
nlp = spacy.load("en_core_web_sm")
spacy_filtered = [token.text for token in nlp(text) if not
token.is_stop]

print("Original Word Count:", len(tokens))
print("After NLTK Stopword Removal:", nltk_filtered)
print("After SpaCy Stopword Removal:", spacy_filtered)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

Original Word Count: 13
After NLTK Stopword Removal: ['example', 'sentence', ',', 'showing',
'effect', 'stopword', 'removal', '.']
After SpaCy Stopword Removal: ['example', 'sentence', ',', 'showing',
'effect', 'stopword', 'removal', '.']
```

```
from nltk.stem import PorterStemmer, LancasterStemmer,
WordNetLemmatizer
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')

words = ["running", "flies", "better", "studies", "wolves", "cities"]

porter = PorterStemmer()
lancaster = LancasterStemmer()
lemmatizer = WordNetLemmatizer()

print("Original Words:", words)
print("Porter:", [porter.stem(w) for w in words])
print("Lancaster:", [lancaster.stem(w) for w in words])
print("Lemmatizer:", [lemmatizer.lemmatize(w) for w in words])
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...

Original Words: ['running', 'flies', 'better', 'studies', 'wolves',
'cities']
Porter: ['run', 'fli', 'better', 'studi', 'wolv', 'citi']
Lancaster: ['run', 'fli', 'bet', 'study', 'wolv', 'city']
Lemmatizer: ['running', 'fly', 'better', 'study', 'wolf', 'city']
```

```
import re
import string

text = "RT @user123!!! The PRICE of Bitcoin hit $30,000 today!!!
#Crypto □□"


text_lower = text.lower()


cleaned = re.sub(r"@\w+|#\w+|http\S+|[^a-zA-Z\s]", "", text_lower)

cleaned = re.sub(r"\s+", " ", cleaned).strip()

print("Original:", text)
print("Cleaned:", cleaned)
```

```
Original: RT @user123!!! The PRICE of Bitcoin hit $30,000 today!!!
#Crypto □□
Cleaned: rt the price of bitcoin hit today
```

```
gold = [('The', 'DT'), ('dog', 'NN'), ('chased', 'VBD'), ('the',
'DT'), ('cat', 'NN')]
text = "The dog chased the cat."


tokens = nltk.word_tokenize(text)
nltk_tags = nltk.pos_tag(tokens)


doc = nlp(text)
spacy_tags = [(token.text, token.tag_) for token in doc]


def compute_accuracy(pred, gold):
    correct = sum(1 for p, g in zip(pred, gold) if p[1] == g[1])
    return correct / len(gold)

print("NLTK Accuracy:", compute_accuracy(nltk_tags, gold))
print("SpaCy Accuracy:", compute_accuracy(spacy_tags, gold))

from nltk import RegexpTagger

patterns = [
    (r'.*ing$', 'VBG'),
    (r'.*ed$', 'VBD'),
    (r'.*es$', 'VBZ'),
    (r'^Ravi$', 'NNP'),
    (r'cricket|TV', 'NN'),
    (r'the|and|a|daily', 'DT'),
    (r'.*', 'NN')
```

```
]

tagger = RegexpTagger(patterns)
sentence = ['Ravi', 'plays', 'cricket', 'and', 'watches', 'TV',
'daily']
print("Rule-Based Tags:", tagger.tag(sentence))
Expected Output:
Rule-Based Tags: [('Ravi', 'NNP'), ('plays', 'VBZ'), ('cricket',
'NN'),
 ('and', 'DT'), ('watches', 'VBZ'), ('TV', 'NN'), ('daily', 'DT')]

from nltk.corpus import brown
from nltk.tag import hmm
nltk.download('brown')
nltk.download('universal_tagset')


train_data = brown.tagged_sents(categories='news', tagset='universal')
[:500]


trainer = hmm.HiddenMarkovModelTrainer()
hmm_tagger = trainer.train_supervised(train_data)

sentence = "The quick brown fox jumps over the lazy dog".split()
print("HMM Tags:", hmm_tagger.tag(sentence))
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.

HMM Tags: [('The', 'DET'), ('quick', 'DET'), ('brown', 'DET'), ('fox',
'DET'), ('jumps', 'DET'), ('over', 'DET'), ('the', 'DET'), ('lazy',
'DET'), ('dog', 'DET')]

/usr/local/lib/python3.11/dist-packages/nltk/tag/hmm.py:333:
RuntimeWarning: overflow encountered in cast
  X[i, j] = self._transitions[si].logprob(self._states[j])
/usr/local/lib/python3.11/dist-packages/nltk/tag/hmm.py:335:
RuntimeWarning: overflow encountered in cast
  O[i, k] = self._output_logprob(si, self._symbols[k])
/usr/local/lib/python3.11/dist-packages/nltk/tag/hmm.py:331:
RuntimeWarning: overflow encountered in cast
  P[i] = self._priors.logprob(si)
/usr/local/lib/python3.11/dist-packages/nltk/tag/hmm.py:363:
RuntimeWarning: overflow encountered in cast
  O[i, k] = self._output_logprob(si, self._symbols[k])
```

```
from transformers import pipeline
```

```
nlp_pipeline = pipeline("token-classification", model="dslim/bert-base-NER")

text = "Elon Musk founded SpaceX in 2002."
output = nlp_pipeline(text)

for item in output:
    print(item)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Secrets can only be fetched when running from the Colab UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub (https://github.com/huggingface/huggingface_hub/issues/new).
  warnings.warn(
```


```
{"model_id":"93cdd878eb774e739427d41907c6df6b","version_major":2,"version_minor":0}
```

```
{"model_id":"05c0edb5f00a41c7b780d6e150f5c58e","version_major":2,"version_minor":0}
```


```
Some weights of the model checkpoint at dslim/bert-base-NER were not used when initializing BertForTokenClassification:
['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```


```
{"model_id":"d1e2514f827e4fe099a02c1fdb7225cb","version_major":2,"version_minor":0}
```

```
{"model_id":"b5ac7600803940adb568c180a134517a","version_major":2,"version_minor":0}
```

```
{"model_id":"f447e037e7104bbeb23b3ae3ad47a074","version_major":2,"version_minor":0}
```

```
{"model_id":"d285f77fbba14152a90a525a0f46f34c","version_major":2,"version_minor":0}
```

```
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:175
0: FutureWarning: `encoder_attention_mask` is deprecated and will be
removed in version 4.55.0 for `BertSdpaSelfAttention.forward`.
  return forward_call(*args, **kwargs)

{'entity': 'B-ORG', 'score': np.float32(0.7459889), 'index': 1,
'word': 'El', 'start': 0, 'end': 2}
{'entity': 'I-ORG', 'score': np.float32(0.7533793), 'index': 2,
'word': '##on', 'start': 2, 'end': 4}
{'entity': 'I-PER', 'score': np.float32(0.7262221), 'index': 3,
'word': 'Mu', 'start': 5, 'end': 7}
{'entity': 'I-ORG', 'score': np.float32(0.6327566), 'index': 4,
'word': '##sk', 'start': 7, 'end': 9}
{'entity': 'B-ORG', 'score': np.float32(0.9993462), 'index': 6,
'word': 'Space', 'start': 18, 'end': 23}
{'entity': 'I-ORG', 'score': np.float32(0.99909484), 'index': 7,
'word': '##X', 'start': 23, 'end': 24}

import nltk, spacy
from nltk.corpus import stopwords
from transformers import BertTokenizer

nltk.download('punkt')
nltk.download('stopwords')
nlp = spacy.load("en_core_web_sm")

text1 = "Artificial Intelligence is revolutionizing the world."
text2 = "This is an example sentence, showing the effect of stopword
removal."

# Tokenization
nltk_tokens = nltk.word_tokenize(text1)
spacy_tokens = [t.text for t in nlp(text1)]
char_tokens = list(text1)
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
subword_tokens = tokenizer.tokenize(text1)

# Stopword removal
tokens2 = nltk.word_tokenize(text2)
nltk_filtered = [w for w in tokens2 if w.lower() not in
stopwords.words('english')]
spacy_filtered = [t.text for t in nlp(text2) if not t.is_stop]

print("NLTK Tokens:", nltk_tokens)
print("SpaCy Tokens:", spacy_tokens)
print("Char Tokens (first 15):", char_tokens[:15])
print("Subword Tokens:", subword_tokens)
print("NLTK Stopword Removal:", nltk_filtered)
print("SpaCy Stopword Removal:", spacy_filtered)
```

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

NLTK Tokens: ['Artificial', 'Intelligence', 'is', 'revolutionizing',
'the', 'world', '.']
SpaCy Tokens: ['Artificial', 'Intelligence', 'is', 'revolutionizing',
'the', 'world', '.']
Char Tokens (first 15): ['A', 'r', 't', 'i', 'f', 'i', 'c', 'i', 'a',
'l', ' ', 'I', 'n', 't', 'e']
Subword Tokens: ['artificial', 'intelligence', 'is', 'revolution',
'##izing', 'the', 'world', '.']
NLTK Stopword Removal: ['example', 'sentence', ',', 'showing',
'effect', 'stopword', 'removal', '.']
SpaCy Stopword Removal: ['example', 'sentence', ',', 'showing',
'effect', 'stopword', 'removal', '.']
```

```python
import nltk, spacy
nltk.download('averaged_perceptron_tagger')

text1 = "John loves eating pizza while Mary reads books in the
library."
text2 = "Barack Obama was born in Hawaii."

# POS tagging
tokens = nltk.word_tokenize(text1)
nltk_tags = nltk.pos_tag(tokens)

nlp = spacy.load("en_core_web_sm")
doc = nlp(text1)
spacy_tags = [(t.text, t.pos_) for t in doc]

# Chunking
tokens2 = nltk.word_tokenize(text2)
tags2 = nltk.pos_tag(tokens2)
grammar = "NP: {<DT>?<JJ>*<NNP>+}"
chunk_parser = nltk.RegexpParser(grammar)
tree = chunk_parser.parse(tags2)

print("NLTK Tags:", nltk_tags)
print("SpaCy Tags:", spacy_tags)
print("Noun Phrase Chunks:", [subtree.leaves() for subtree in
tree.subtrees() if subtree.label()=='NP'])
```