```
#1
import spacy
# Load English tokenizer, tagger, parser, NER, and word vectors
nlp = spacy.load("en_core_web_sm")
# Sample text for analysis
text = "Natural Language Processing is a fascinating field of study."
# Process the text with spaCy
doc = nlp(text)
# Extracting tokens and lemmatization
tokens = [token.text for token in doc]
lemmas = [token.lemma_ for token in doc]
print("Tokens:", tokens)
print("Lemmas:", lemmas)
# Dependency parsing
print("\nDependency Parsing:")
for token in doc:
 print(token.text, token.dep_, token.head.text, token.head.pos_,
 [child for child in token.children])
```

```
Tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field', 'of', 'study', '.']
Lemmas: ['Natural', 'Language', 'Processing', 'be', 'a', 'fascinating', 'field', 'of', 'study', '.']

Dependency Parsing:
Natural compound Language PROPN []
Language compound Processing PROPN [Natural]
Processing nsubj is AUX [Language]
is ROOT is AUX [Processing, field, .]
a det field NOUN []
fascinating amod field NOUN []
field attr is AUX [a, fascinating, of]
of prep field NOUN [study]
study pobj of ADP []
. punct is AUX []
```

```
#1 case study
import spacy
# Load English tokenizer, tagger, parser, NER, and word vectors
nlp = spacy.load("en_core_web_sm")
# Sample customer feedback data
customer_feedback = [
 "The product is amazing! I love the quality.",
 "The customer service was terrible, very disappointed.",
 "Great experience overall, highly recommended.",
 "The delivery was late, very frustrating."
 ]
def analyze_feedback(feedback):
    for idx, text in enumerate(feedback, start=1):
        print(f"\nAnalyzing Feedback {idx}: '{text}'")
        doc = nlp(text)
    tokens = [token.text for token in doc]
    lemmas = [token.lemma_ for token in doc]
    print("Tokens:", tokens)
    print("Lemmas:", lemmas)
    print("\nDependency Parsing:")
    for token in doc:
        print(token.text, token.dep_, token.head.text, token.head.pos_,
              [child for child in token.children])
if __name__ == "__main__":
 analyze_feedback(customer_feedback)
```

```
Analyzing Feedback 1: 'The product is amazing! I love the quality.'

Analyzing Feedback 2: 'The customer service was terrible, very disappointed.'

Analyzing Feedback 3: 'Great experience overall, highly recommended.'

Analyzing Feedback 4: 'The delivery was late, very frustrating.'
Tokens: ['The', 'delivery', 'was', 'late', ',', 'very', 'frustrating', '.']
Lemmas: ['the', 'delivery', 'be', 'late', ',', 'very', 'frustrating', '.']

Dependency Parsing:
The det delivery NOUN []
delivery nsubj was AUX [The]
was ROOT was AUX [delivery, frustrating, .]
late advmod frustrating ADJ []
, punct frustrating ADJ []
very advmod frustrating ADJ []
frustrating acomp was AUX [late, ,, very]
. punct was AUX []
```

```
#2
import nltk
import random

nltk.download('punkt')
nltk.download('gutenberg')

words = nltk.corpus.gutenberg.words()

bigrams = list(nltk.bigrams(words))

starting_word = "the"
generated_text = [starting_word]

for _ in range(20):

 possible_words = [word2 for (word1, word2) in bigrams if word1.lower() == generated_text[-1].lower()]


 next_word = random.choice(possible_words)
 generated_text.append(next_word)

print(' '.join(generated_text))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
the mast head and the son , " If you can afford it doesn ' s eye spare them more step
```

```
#2 Case study
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
class EmailAutocompleteSystem:
    def __init__(self):
        self.model_name = "gpt2"
        self.tokenizer = GPT2Tokenizer.from_pretrained(self.model_name)
        self.model = GPT2LMHeadModel.from_pretrained(self.model_name)
    def generate_suggestions(self, user_input, context):
        input_text = f"{context} {user_input}"
        input_ids = self.tokenizer.encode(input_text, return_tensors="pt")
        with torch.no_grad():
            output = self.model.generate(input_ids, max_length=50, num_return_sequences=1,no_repeat_ngram_size=2)
        generated_text = self.tokenizer.decode(output[0], skip_special_tokens=True)
        suggestions = generated_text.split()[len(user_input.split()):]
        return suggestions

if __name__ == "__main__":
    autocomplete_system = EmailAutocompleteSystem()
    email_context = "Subject: Discussing Project Proposal\nHi [Recipient],"
    while True:
        user_input = input("Enter your sentence (type 'exit' to end): ")
        if user_input.lower() == 'exit':
            break
        suggestions = autocomplete_system.generate_suggestions(user_input, email_context)
        if suggestions:
            print("Autocomplete Suggestions:", suggestions)
        else:
            print("No suggestions available.")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarni
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mc
  warnings.warn(
tokenizer_config.json:  100%          26.0/26.0  [00:00<00:00,  636B/s]
vocab.json:  100%                     1.04M/1.04M  [00:00<00:00,  6.15MB/s]
merges.txt:  100%                     456k/456k  [00:00<00:00,  2.20MB/s]
tokenizer.json:  100%                 1.36M/1.36M  [00:00<00:00,  8.90MB/s]
config.json:  100%                    665/665  [00:00<00:00,  12.9kB/s]
model.safetensors:  100%              548M/548M  [00:09<00:00,  41.4MB/s]
generation_config.json:  100%         124/124  [00:00<00:00,  588B/s]
```

```
#3
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
# Load the 20 Newsgroups dataset
categories = ['sci.med', 'sci.space', 'comp.graphics', 'talk.politics.mideast']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
# Split the data into training and testing sets
X_train = newsgroups_train.data
X_test = newsgroups_test.data
y_train = newsgroups_train.target
y_test = newsgroups_test.target
# Create a pipeline with TF-IDF vectorizer and LinearSVC classifier
model = make_pipeline(
 TfidfVectorizer(),
 LinearSVC()
)
# Train the model
model.fit(X_train, y_train)
# Predict labels for the test set
predictions = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, predictions))
```

```
Accuracy: 0.9504823151125402

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       389
           1       0.96      0.91      0.94       396
           2       0.98      0.94      0.96       394
           3       0.98      0.98      0.98       376

    accuracy                           0.95      1555
   macro avg       0.95      0.95      0.95      1555
weighted avg       0.95      0.95      0.95      1555
```

```
#3 Case study
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
# Load the 20 Newsgroups dataset as a proxy for customer support emails
newsgroups = fetch_20newsgroups(subset='all', categories=['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'rec.autos', 'rec.motorcy
# Prepare data and target labels
X = newsgroups.data
y = newsgroups.target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=10000)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
# Train the LinearSVC classifier
classifier = LinearSVC()
classifier.fit(X_train, y_train)
# Predict labels for the test set
predictions = classifier.predict(X_test)
# Evaluate the classifier
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, predictions, target_names=newsgroups.target_names))
```

```
Accuracy: 0.9389623601220752

Classification Report:
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.92      0.91      0.91       212
   comp.sys.mac.hardware       0.94      0.93      0.94       198
               rec.autos       0.97      0.93      0.95       179
         rec.motorcycles       0.96      0.99      0.97       205
         sci.electronics       0.92      0.93      0.92       189

                accuracy                           0.94       983
               macro avg       0.94      0.94      0.94       983
            weighted avg       0.94      0.94      0.94       983
```

```
#4
# Install necessary libraries
!pip install gensim
!pip install nltk
# Import required libraries
import gensim.downloader as api
from nltk.tokenize import word_tokenize
# Download pre-trained word vectors (Word2Vec)
word_vectors = api.load("word2vec-google-news-300")
# Sample sentences
sentences = [
"Natural language processing is a challenging but fascinating field.",
"Word embeddings capture semantic meanings of words in a vector space."
]
# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
# Perform semantic analysis using pre-trained word vectors
for tokenized_sentence in tokenized_sentences:
  for word in tokenized_sentence:
    if word in word_vectors:
      similar_words = word_vectors.most_similar(word)
      print(f"Words similar to '{word}': {similar_words}")
    else:
      print(f"'{word}' is not in the pre-trained Word2Vec model.")
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.25.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.11.4)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim) (6.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
Words similar to 'natural': [('Splittorff_lacked', 0.636509358882904), ('Natural', 0.58078932762146), ('Mike_Taugher_covers', 0.577
Words similar to 'language': [('langauge', 0.7476695775985718), ('Language', 0.6695356369018555), ('languages', 0.6341332197189331)
Words similar to 'processing': [('Processing', 0.7285515666007996), ('processed', 0.6519132852554321), ('processor', 0.636760413646
Words similar to 'is': [('was', 0.6549733281135559), ("isn'ta", 0.6439523100852966), ('seems', 0.634029746055603), ('Is', 0.608596
'a' is not in the pre-trained Word2Vec model.
Words similar to 'challenging': [('difficult', 0.6388775110244751), ('challenge', 0.5953003764152527), ('daunting', 0.569800615310
Words similar to 'but': [('although', 0.8104525804519653), ('though', 0.7285684943199158), ('because', 0.7225914597511292), ('so',
Words similar to 'fascinating': [('interesting', 0.7623067498207092), ('intriguing', 0.7245113253593445), ('enlightening', 0.664425
Words similar to 'field': [('fields', 0.5582526326179504), ('fi_eld', 0.5188260078430176), ('Keith_Toogood', 0.49749255180358887),
'.' is not in the pre-trained Word2Vec model.
Words similar to 'word': [('phrase', 0.6777030825614929), ('words', 0.5864380598068237), ('verb', 0.5517287254333496), ('Word', 0.5
'embeddings' is not in the pre-trained Word2Vec model.
Words similar to 'capture': [('capturing', 0.7563897371292114), ('captured', 0.7155306935310364), ('captures', 0.6099075078964233),
Words similar to 'semantic': [('semantics', 0.6644964814186096), ('Semantic', 0.6464474201202393), ('contextual', 0.590912759304046
Words similar to 'meanings': [('grammatical_constructions', 0.594986081123352), ('idioms', 0.5938195586204529), ('connotations', 0.
'of' is not in the pre-trained Word2Vec model.
Words similar to 'words': [('phrases', 0.7100036144256592), ('phrase', 0.6408688426017761), ('Words', 0.6160537600517273), ('word',
Words similar to 'in': [('inthe', 0.5891957879066467), ('where', 0.5662435293197632), ('the', 0.5429296493530273), ('In', 0.541511
'a' is not in the pre-trained Word2Vec model.
Words similar to 'vector': [('vectors', 0.750322163105011), ('adeno_associated_viral_AAV', 0.5999537110328674), ('bitmap_graphics',
Words similar to 'space': [('spaces', 0.6570690870285034), ('music_concept_ShockHound', 0.5850345492362976), ('Shuttle_docks', 0.55
'.' is not in the pre-trained Word2Vec model.
```

```
#4 case study
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
# Initialize NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
# Function to perform semantic analysis
def semantic_analysis(text):
  tokens = word_tokenize(text)
  stop_words = set(stopwords.words('english'))
  filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
  lemmatizer = WordNetLemmatizer()
  lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
  synonyms = set()
  for token in lemmatized_tokens:
    for syn in wordnet.synsets(token):
      for lemma in syn.lemmas():
        synonyms.add(lemma.name())
  return list(synonyms)
# Example customer queries
customer_queries = [
"I received a damaged product. Can I get a refund?",
"I'm having trouble accessing my account.",
"How can I track my order status?",
"The item I received doesn't match the description.",
"Is there a discount available for bulk orders?"
]
# Semantic analysis for each query
for query in customer_queries:
  print("Customer Query:", query)
  synonyms = semantic_analysis(query)
  print("Semantic Analysis (Synonyms):", synonyms)
  print("\n")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Customer Query: I received a damaged product. Can I get a refund?
Semantic Analysis (Synonyms): ['generate', 'fuck_off', 'damaged', 'bewilder', 'get', 'stick', 'draw', 'pick_up', 'take_in', 'develop


Customer Query: I'm having trouble accessing my account.
Semantic Analysis (Synonyms): ['disturb', 'news_report', 'calculate', 'answer_for', 'accounting', 'invoice', 'account_statement', 'b


Customer Query: How can I track my order status?
Semantic Analysis (Synonyms): ['racetrack', 'position', 'parliamentary_procedure', 'society', 'gild', 'tell', 'order_of_magnitude',


Customer Query: The item I received doesn't match the description.
Semantic Analysis (Synonyms): ['oppose', 'invite', 'received', 'couple', 'agree', 'check', 'receive', 'get', 'friction_match', 'meet


Customer Query: Is there a discount available for bulk orders?
Semantic Analysis (Synonyms): ['orderliness', 'dictate', 'ordination', 'edict', 'bank_discount', 'parliamentary_procedure', 'set_up
```

```
#5
# Install necessary libraries
!pip install scikit-learn
!pip install nltk
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import movie_reviews # Sample dataset from NLTK
# Download NLTK resources (run only once if not downloaded)
import nltk
nltk.download('movie_reviews')
# Load the movie_reviews dataset
documents = [(list(movie_reviews.words(fileid)), category)
for category in movie_reviews.categories()
for fileid in movie_reviews.fileids(category)]

# Convert data to DataFrame
df = pd.DataFrame(documents, columns=['text', 'sentiment'])
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['sentiment'], test_size=0.2,
random_state=42)
# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
# Fit and transform the training data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train.apply(' '.join))
# Initialize SVM classifier
svm_classifier = SVC(kernel='linear')
# Train the classifier
svm_classifier.fit(X_train_tfidf, y_train)
# Transform the test data
X_test_tfidf = tfidf_vectorizer.transform(X_test.apply(' '.join))
# Predict on the test data
y_pred = svm_classifier.predict(X_test_tfidf)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
# Display classification report
print(classification_report(y_test, y_pred))
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
Accuracy: 0.84
              precision    recall  f1-score   support

         neg       0.83      0.85      0.84       199
         pos       0.85      0.82      0.84       201

    accuracy                           0.84       400
   macro avg       0.84      0.84      0.84       400
weighted avg       0.84      0.84      0.84       400
```

```
#5 case study
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# Download NLTK resources (only required once)
nltk.download('vader_lexicon')
# Sample reviews
reviews = [
"This product is amazing! I love it.",
"The product was good, but the packaging was damaged.",
"Very disappointing experience. Would not recommend.",
"Neutral feedback on the product.",
]
# Initialize Sentiment Intensity Analyzer
sid = SentimentIntensityAnalyzer()
# Analyze sentiment for each review
for review in reviews:
  print("Review:", review)
  scores = sid.polarity_scores(review)
  print("Sentiment:", end=' ')
  if scores['compound'] > 0.05:
    print("Positive")
  elif scores['compound'] < -0.05:
    print("Negative")
  else:
    print("Neutral")
print()
```

```
Review: This product is amazing! I love it.
Sentiment: Positive
Review: The product was good, but the packaging was damaged.
Sentiment: Negative
Review: Very disappointing experience. Would not recommend.
Sentiment: Negative
Review: Neutral feedback on the product.
Sentiment: Neutral

[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

```
#6
# Install NLTK (if not already installed)
!pip install nltk
# Import necessary libraries
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
# Sample text for POS tagging
text = "Parts of speech tagging helps to understand the function of each word in a sentence."
# Tokenize the text into words
tokens = nltk.word_tokenize(text)
# Perform POS tagging
pos_tags = nltk.pos_tag(tokens)
# Display the POS tags
print("POS tags:", pos_tags)
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
POS tags: [('Parts', 'NNS'), ('of', 'IN'), ('speech', 'NN'), ('tagging', 'VBG'), ('helps', 'NNS'), ('to', 'TO'), ('understand', 'VB
#6 Case study
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
# Download NLTK resources (if not already downloaded)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
def pos_tagging(text):
  sentences = sent_tokenize(text)
  tagged_tokens = []
  for sentence in sentences:
    tokens = word_tokenize(sentence)
    tagged_tokens.extend(nltk.pos_tag(tokens))
  return tagged_tokens
def main():
  article_text = """Manchester United secured a 3-1 victory over Chelsea in yesterday's match.
  Goals from Rashford, Greenwood, and Fernandes sealed the win for United.
  Chelsea's only goal came from Pulisic in the first half.
  The victory boosts United's chances in the Premier League title race.
  """
  tagged_tokens = pos_tagging(article_text)
  print("Original Article Text:\n", article_text)
  print("\nParts of Speech Tagging:")
  for token, pos_tag in tagged_tokens:
    print(f"{token}: {pos_tag}")
if __name__ == "__main__":
  main()
```

```
Original Article Text:
 Manchester United secured a 3-1 victory over Chelsea in yesterday's match.
  Goals from Rashford, Greenwood, and Fernandes sealed the win for United.
  Chelsea's only goal came from Pulisic in the first half.
  The victory boosts United's chances in the Premier League title race.


Parts of Speech Tagging:
Manchester: NNP
United: NNP
secured: VBD
a: DT
3-1: JJ
victory: NN
over: IN
Chelsea: NNP
in: IN
yesterday: NN
's: POS
match: NN
.: .
Goals: NNS
from: IN
Rashford: NNP
,: ,
Greenwood: NNP
,: ,
and: CC
Fernandes: NNP
sealed: VBD
the: DT
win: NN
for: IN
United: NNP
.: .
Chelsea: NN
's: POS
only: JJ
goal: NN
came: VBD
from: IN
Pulisic: NNP
in: IN
the: DT
first: JJ
half: NN
.: .
The: DT
victory: NN
boosts: VBZ
United: NNP
's: POS
chances: NNS
in: IN
the: DT
Premier: NNP
League: NNP
title: NN
```

```
#7
!pip install nltk
import nltk
from nltk import RegexpParser
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
# Download NLTK resources (run only once if not downloaded)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
# Sample sentence
sentence = "The quick brown fox jumps over the lazy dog"
# Tokenize the sentence
tokens = word_tokenize(sentence)
# POS tagging
tagged = pos_tag(tokens)
# Define a chunk grammar using regular expressions
# NP (noun phrase) chunking: "NP: {<DT>?<JJ>*<NN>}"
# This grammar captures optional determiner (DT), adjectives (JJ), and nouns (NN) as a noun phrase
chunk_grammar = r"""
 NP: {<DT>?<JJ>*<NN>}
"""
# Create a chunk parser with the defined grammar
chunk_parser = RegexpParser(chunk_grammar)
# Parse the tagged sentence to extract chunks
chunks = chunk_parser.parse(tagged)
# Display the chunks
for subtree in chunks.subtrees():
 if subtree.label() == 'NP':
  print(subtree)
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
(NP The/DT quick/JJ brown/NN)
```