1

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Natural Language Processing is a fascinating field of study."
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])
```

1B

```
import spacy
nlp = spacy.load("en_core_web_sm")
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)
```

2

```
!pip install nltk
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))
```

2B

```
!pip install transformers
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.")
```

3

```
!pip install scikit-learn
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
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)
X_train = newsgroups_train.data
X_test = newsgroups_test.data
y_train = newsgroups_train.target
y_test = newsgroups_test.target
model = make_pipeline(
TfidfVectorizer(),
LinearSVC()
)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, predictions))
```

3B

```
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

newsgroups = fetch_20newsgroups(subset='all', categories=['comp.sys.ibm.pc.hardware',

'comp.sys.mac.hardware', 'rec.autos', 'rec.motorcycles', 'sci.electronics'])

X = newsgroups.data

y = newsgroups.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer(stop_words='english', max_features=10000)

X_train = vectorizer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)

classifier = LinearSVC()

classifier.fit(X_train, y_train)

predictions = classifier.predict(X_test)

accuracy = accuracy_score(y_test, predictions)

print("Accuracy:", accuracy)

print("\nClassification Report:")

print(classification_report(y_test, predictions, target_names=newsgroups.target_names))
```

4

```
!pip install gensim

!pip install nltk

import gensim.downloader as api

from nltk.tokenize import word_tokenize

word_vectors = api.load("word2vec-google-news-300")

sentences = [

"Natural language processing is a challenging but fascinating field.",

"Word embeddings capture semantic meanings of words in a vector space."

]


tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

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.")
```

4B

```
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

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)
```

```
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?"

]

for query in customer_queries:

print("Customer Query:", query)

synonyms = semantic_analysis(query)

print("Semantic Analysis (Synonyms):", synonyms)

print("\n")
```

5

```
!pip install scikit-learn

!pip install nltk

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

import nltk

nltk.download('movie_reviews')

documents = [(list(movie_reviews.words(fileid)), category)

for category in movie_reviews.categories()

for fileid in movie_reviews.fileids(category)]


df = pd.DataFrame(documents, columns=['text', 'sentiment'])
```

```
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['sentiment'], test_size=0.2,
random_state=42)
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train.apply(' '.join))
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_tfidf, y_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test.apply(' '.join))
y_pred = svm_classifier.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
```

5B

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
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.",
]
sid = SentimentIntensityAnalyzer()
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()
```

6

```
!pip install nltk

import nltk

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

text = "Parts of speech tagging helps to understand the function of each word in a sentence."

tokens = nltk.word_tokenize(text)

pos_tags = nltk.pos_tag(tokens)

print("POS tags:", pos_tags)
```

6B

```
import nltk

from nltk.tokenize import word_tokenize, sent_tokenize


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()
```

7

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

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()
```

7B

```
import nltk
import os

nltk.data.path.append("/usr/local/share/nltk_data")

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

text = "The quick brown fox jumps over the lazy dog."

words = nltk.word_tokenize(text)

pos_tags = nltk.pos_tag(words)

chunk_grammar = r"""
    NP: {<DT>?<JJ>*<NN>}  # Chunk sequences of DT, JJ, NN
"""

chunk_parser = nltk.RegexpParser(chunk_grammar)

chunked_text = chunk_parser.parse(pos_tags)
```

```
noun_phrases = []

for subtree in chunked_text.subtrees(filter=lambda t: t.label() ==

'NP'):

    noun_phrases.append(' '.join(word for word, tag in

subtree.leaves()))


print("Original Text:", text)

print("Noun Phrases:")

for phrase in noun_phrases:

print("-", phrase)
```