if39p8jhb

April 7, 2025

0.0.1 1. Define Natural Language Processing (NLP). Provide three real-world applications of NLP and explain how they impact society.

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful. It combines computational linguistics, machine learning, and deep learning techniques to process and analyze text or speech data, allowing machines to mimic human-like communication. Three Real-World Applications of NLP and Their Societal Impact: 1. Virtual Assistants (e.g., Siri, Alexa, Google Assistant)
* How it works: NLP powers virtual assistants by interpreting spoken or typed commands, extracting intent, and providing relevant responses or actions, such as setting reminders or answering questions.
* Societal Impact: These tools enhance accessibility, helping people with disabilities interact with technology more easily. They save time, boost productivity, and have become integral to daily life, though they also raise privacy concerns due to constant listening capabilities. 2. Sentiment Analysis in Social Media (e.g., Brand Monitoring)
* How it works: NLP algorithms analyze posts, comments, and reviews to determine public sentiment (positive, negative, or neutral) toward brands, products, or events.
* Societal Impact: Businesses use this to adapt marketing strategies and improve customer satisfaction, fostering better consumer relationships. However, it can also amplify echo chambers or misinterpret sarcasm, influencing public perception inaccurately. 3. Machine Translation (e.g., Google Translate)
* How it works: NLP translates text or speech from one language to another by understanding syntax, semantics, and cultural nuances.
* Societal Impact: It breaks down language barriers, facilitating global communication, education, and commerce. It empowers immigrants and travelers but can sometimes produce errors, leading to misunderstandings in critical contexts like healthcare or diplomacy. * NLP's advancements continue to transform how we interact with technology and each other, driving efficiency and connectivity while posing challenges around ethics, accuracy, and data privacy.

0.0.2 2. Explain the following terms and their significance in NLP:

- Tokenization
- Stemming
- Lemmatization

Here's an explanation of tokenization, stemming, and lemmatization, along with their significance in Natural Language Processing (NLP): 1. Tokenization * Definition: Tokenization is the process of breaking down a piece of text (e.g., a sentence or document) into smaller units called "tokens."

These tokens are typically words, but they can also be phrases, sentences, or even individual characters, depending on the task.

- Example: For the sentence "I love running," tokenization might produce: ["I", "love", "running"].
- Significance in NLP: Tokenization is a foundational step in NLP pipelines. It simplifies text into manageable pieces for further analysis, such as counting word frequency or feeding tokens into a machine learning model. Without it, raw text would be too unstructured for computers to process effectively. The choice of tokenization (e.g., word-level vs. subword-level) can also impact model performance in tasks like translation or sentiment analysis.

2. Stemming

- Definition: Stemming is the process of reducing words to their root or base form (stem) by removing suffixes or prefixes, often using heuristic rules. It doesn't always produce a valid word, just a common root.
- Example: Stemming "running," "runner," and "runs" might all result in "run."
- Significance in NLP: Stemming reduces vocabulary size and groups related words together, which is crucial for tasks like search engines or text classification. By treating variations of a word as the same entity, it improves efficiency and recall (finding relevant results). However, its simplicity can lead to errors (e.g., "university" and "universe" both stemming to "univers"), making it less precise than alternatives.

3. Lemmatization

- Definition: Lemmatization is the process of reducing words to their canonical or dictionary form (lemma) by considering the word's meaning and part of speech. Unlike stemming, it ensures the result is a valid word.
- Example: Lemmatizing "running" (verb) becomes "run," while "mice" (noun) becomes "mouse."
- Significance in NLP: Lemmatization provides more accurate normalization than stemming, preserving semantic meaning. This is vital for tasks requiring context, like question answering or chatbot development, where understanding intent is key. Though computationally heavier than stemming, it enhances precision in applications like machine translation or sentiment analysis by maintaining linguistic integrity.

Broader Impact in NLP These techniques are critical preprocessing steps that enable machines to handle the complexity and variability of human language. Tokenization sets the stage, while stemming and lemmatization reduce noise and redundancy, improving model accuracy and efficiency. Their choice depends on the task: stemming for speed in simple applications, lemmatization for precision in nuanced ones. Together, they bridge the gap between messy human text and structured data machines can understand.

3. **What is Part-of-Speech (POS) tagging? Discuss its importance with an example.**

Answer: POS tagging is the process of labeling each word in a sentence with its grammatical category (e.g., noun, verb, adjective). It's crucial for understanding sentence structure and meaning.

Importance: It helps in tasks like text parsing, machine translation, and named entity recognition by providing context.

Example: For "The cat runs fast":

- The (Determiner), cat (Noun), runs (Verb), fast (Adverb).
- This tagging helps a translation system know "runs" is an action, ensuring accurate translation.

0.0.3 4. Create a TextBlob named exercise_blob containing "This is a TextBlob"

[7]:
```
from textblob import TextBlob

exercise_blob = TextBlob("This is a TextBlob")
print(exercise_blob)
```

```
This is a TextBlob
```

0.0.4 5. Write a Python script to perform the following tasks on the given text:

- Tokenize into words and sentences.
- Perform stemming and lemmatization using NLTK or SpaCy.
- Remove stop words.

[14]:
```
from textblob import TextBlob
from nltk.corpus import stopwords
import nltk


nltk.download('stopwords')


text = """Natural Language Processing enables machines to understand and␣
 ↪process human languages.
It is a fascinating field with numerous applications, such as chatbots and␣
 ↪language translation."""


blob = TextBlob(text)

print("Sentences:")
print(blob.sentences)

print("\nWords:")
print(blob.words)


stop_words = set(stopwords.words('english'))
filtered_words = [word for word in blob.words if word.lower() not in stop_words]
print("\nWords after Stop Word Removal:")
print(filtered_words)
```

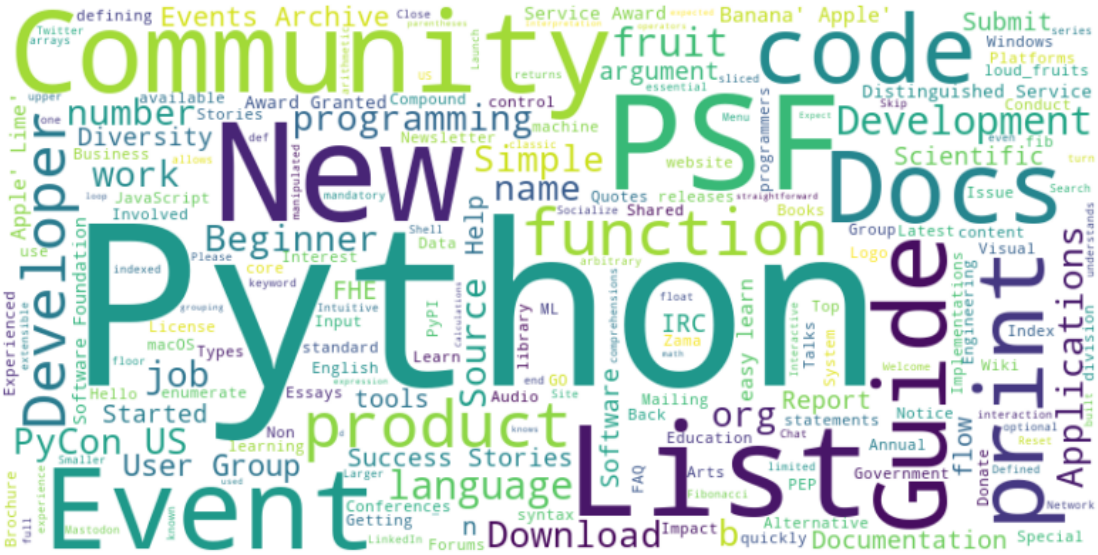
```
lemmatized_words = [word.lemmatize() for word in blob.words]
print("\nLemmatized Words:")
print(lemmatized_words)


from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in blob.words if word.isalpha()]
print("\nStemmed Words:")
print(stemmed_words)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\dpasa\AppData\Roaming\nltk_data…
[nltk_data]   Unzipping corpora\stopwords.zip.
```

```
Sentences:
[Sentence("Natural Language Processing enables machines to understand and
process human languages."), Sentence("It is a fascinating field with numerous
applications, such as chatbots and language translation.")]

Words:
['Natural', 'Language', 'Processing', 'enables', 'machines', 'to', 'understand',
'and', 'process', 'human', 'languages', 'It', 'is', 'a', 'fascinating', 'field',
'with', 'numerous', 'applications', 'such', 'as', 'chatbots', 'and', 'language',
'translation']

Words after Stop Word Removal:
['Natural', 'Language', 'Processing', 'enables', 'machines', 'understand',
'process', 'human', 'languages', 'fascinating', 'field', 'numerous',
'applications', 'chatbots', 'language', 'translation']

Lemmatized Words:
['Natural', 'Language', 'Processing', 'enables', 'machine', 'to', 'understand',
'and', 'process', 'human', 'language', 'It', 'is', 'a', 'fascinating', 'field',
'with', 'numerous', 'application', 'such', 'a', 'chatbots', 'and', 'language',
'translation']

Stemmed Words:
['natur', 'languag', 'process', 'enabl', 'machin', 'to', 'understand', 'and',
'process', 'human', 'languag', 'it', 'is', 'a', 'fascin', 'field', 'with',
'numer', 'applic', 'such', 'as', 'chatbot', 'and', 'languag', 'translat']
```

6. **Web Scraping with the Requests and Beautiful Soup Libraries:** * Use the requests library to download the www.python.org home page's content. * Use the Beautiful Soup library to extract only the text from the page. * Eliminate the stop words in the resulting text, then use the wordcloud module to create a word cloud based on the text.

[4]:
```
import requests
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt


response = requests.get("https://www.python.org")
soup = BeautifulSoup(response.text, 'html.parser')
text = soup.get_text()

stop_words = set(stopwords.words('english'))
words = text.split()
filtered_text = ' '.join([word for word in words if word.lower() not in␣
 ↪stop_words])


wordcloud = WordCloud(width=800, height=400, background_color='white').
 ↪generate(filtered_text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



7. **(Tokenizing Text and Noun Phrases) Using the text from above problem, create a TextBlob, then tokenize it into Sentences and Words, and extract its noun phrases.**

[5]:
```
from textblob import TextBlob

text = "Natural Language Processing enables machines to understand and process␣
 ↪human languages. It is a fascinating field with numerous applications, such␣
 ↪as chatbots and language translation."
blob = TextBlob(text)


print("Sentences:", blob.sentences)
print("Words:", blob.words)
print("Noun Phrases:", blob.noun_phrases)
```

```
Sentences: [Sentence("Natural Language Processing enables machines to understand
and process human languages."), Sentence("It is a fascinating field with
numerous applications, such as chatbots and language translation.")]
Words: ['Natural', 'Language', 'Processing', 'enables', 'machines', 'to',
'understand', 'and', 'process', 'human', 'languages', 'It', 'is', 'a',
'fascinating', 'field', 'with', 'numerous', 'applications', 'such', 'as',
'chatbots', 'and', 'language', 'translation']
Noun Phrases: ['language processing', 'process human languages', 'numerous
applications', 'language translation']
```

8. **(Sentiment of a News Article) Using the techniques in problem no. 5, download a web page for a current news article and create a TextBlob. Display the sentiment for the entire TextBlob and for each Sentence.**

[10]:
```
import requests
from bs4 import BeautifulSoup
from textblob import TextBlob

# India Today article
url = "https://www.indiatoday.in/india/story/
 ↪imf-raises-india-gdp-growth-forecast-to-66-per-cent-in-2024-on-strong-domestic-demand-25190

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36␣
 ↪"
                  "(KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36"
}

response = requests.get(url, headers=headers)

if response.status_code == 200:
    soup = BeautifulSoup(response.content, 'html.parser')


    paragraphs = soup.find_all("p")
    article_text = ' '.join([p.get_text() for p in paragraphs if p.get_text()])
```

```
    blob = TextBlob(article_text)

    print("********* Overall Sentiment:**************")
    print(f"Polarity: {blob.sentiment.polarity}")
    print(f"Subjectivity: {blob.sentiment.subjectivity}\n")

    print("************ Sentiment by Sentence:****************")
    for i, sentence in enumerate(blob.sentences, 1):
        print(f"Sentence {i}: {sentence}")
        print(f"   Polarity: {sentence.sentiment.polarity}")
        print(f"   Subjectivity: {sentence.sentiment.subjectivity}\n")

else:
    print(f" Failed to retrieve article. Status code: {response.status_code}")
```

```
********* Overall Sentiment:**************
Polarity: 0.12954545454545455
Subjectivity: 0.44999999999999996

************ Sentiment by Sentence:****************
Sentence 1: Listen to Story Arun Govil, the veteran actor famed for his role of
Lord Ram in the popular TV show Ramayan, was on Sunday fielded by the BJP as a
candidate from his hometown Meerut in Uttar Pradesh for the upcoming Lok Sabha
elections.
   Polarity: 0.3
   Subjectivity: 0.45

Sentence 2: Govil (66), who was among the 111 people named in the BJP's fifth
list of candidates, replaces three-time MP Rajendra Agarwal who has been holding
the Meerut seat since 2004.
   Polarity: 0.0
   Subjectivity: 0.0

Sentence 3: In a tweet on X, Govil thanked Prime Minister Narendra Modi for
entrusting him with such a "big responsibility" and said he will make efforts to
live up to the expectations of the people if he is elected.
   Polarity: 0.04545454545454545
   Subjectivity: 0.3666666666666667

Sentence 4: "Heartfelt gratitude to Shri Narendra Modi ji and the selection
committee who have given me such a big responsibility by making me the MP
candidate of Meerut.
   Polarity: 0.0
   Subjectivity: 0.5333333333333333

Sentence 5: I will make every effort to fully live up to the trust of the BJP
```

and the expectations of the people.
   Polarity: 0.13636363636363635
   Subjectivity: 0.5

Sentence 6: Jai Shri Ram," he wrote in Hindi.
   Polarity: 0.0
   Subjectivity: 0.0

Sentence 7: Govil was amongst the celebrities who had attended the grand consecration ceremony of the Ram Temple in Ayodhya in January.
   Polarity: 0.5
   Subjectivity: 1.0

Sentence 8: The veteran actor joined the BJP in 2021 and praised PM Modi for his government's policies and changing the political narratives.
   Polarity: 0.0
   Subjectivity: 0.05

Sentence 9: Notably, Govil had essayed the role of PM Modi in the latest released film 'Article 370', which stars Yami Gautam.
   Polarity: 0.5
   Subjectivity: 0.7

Sentence 10: Hailing from Meerut, Govil enrolled for a BSc before he became an actor.
   Polarity: 0.0
   Subjectivity: 0.0

Sentence 11: His debut film, Paheli, was released in 1977.
   Polarity: 0.0
   Subjectivity: 0.0

Sentence 12: Govil became a household name after he played the role of Lord Ram in Ramanand Sagar's TV adaption of Ramayan in the late 1980s.
   Polarity: -0.3
   Subjectivity: 0.6

Sentence 13: He then went on to act in several television shows and also worked in a number of Odia, Telugu, Bhojpuri and Braj Bhasha films.Published By: Prateek ChakrabortyPublished On: Mar 24, 2024ALSO READ | BJP drops Varun Gandhi from Pilibhit, retains mother Maneka in SultanpurALSO READ | BJP leader VK Singh to not contest 2024 Lok Sabha pollsMust Watch
   Polarity: 0.0
   Subjectivity: 0.0

0.0.5 9. (Sentiment of a News Article with the NaiveBayesAnalyzer) Repeat the previous exercise but use the NaiveBayesAnalyzer for sentiment analysis.

[12]:
```
import requests
from bs4 import BeautifulSoup
from textblob import TextBlob
from textblob.sentiments import NaiveBayesAnalyzer
import nltk


# nltk.download('punkt')
# nltk.download('movie_reviews')


url = "https://www.indiatoday.in/india/story/
 ↪imf-raises-india-gdp-growth-forecast-to-66-per-cent-in-2024-on-strong-domestic-demand-25190

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36␣
 ↪"
                  "(KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36"
}

response = requests.get(url, headers=headers)

if response.status_code == 200:
    soup = BeautifulSoup(response.content, 'html.parser')


    paragraphs = soup.find_all("p")
    article_text = ' '.join([p.get_text() for p in paragraphs if p.get_text()])


    blob = TextBlob(article_text, analyzer=NaiveBayesAnalyzer())

    print(" Overall Sentiment:")
    print(f"Classification: {blob.sentiment.classification}")
    print(f"Pos Probability: {blob.sentiment.p_pos}")
    print(f"Neg Probability: {blob.sentiment.p_neg}\n")

    print(" Sentiment by Sentence:")
    for i, sentence in enumerate(blob.sentences, 1):
        sentiment = sentence.sentiment
        print(f"Sentence {i}: {sentence}")
        print(f"   Classification: {sentiment.classification}")
        print(f"   Pos: {sentiment.p_pos:.4f}, Neg: {sentiment.p_neg:.4f}\n")
```

```
else:
    print(f" Failed to retrieve article. Status code: {response.status_code}")
```

```
 Overall Sentiment:
Classification: pos
Pos Probability: 0.9999996372767355
Neg Probability: 3.6272323896104994e-07

 Sentiment by Sentence:
Sentence 1: Listen to Story Arun Govil, the veteran actor famed for his role of
Lord Ram in the popular TV show Ramayan, was on Sunday fielded by the BJP as a
candidate from his hometown Meerut in Uttar Pradesh for the upcoming Lok Sabha
elections.
   Classification: pos
   Pos: 0.8445, Neg: 0.1555

Sentence 2: Govil (66), who was among the 111 people named in the BJP's fifth
list of candidates, replaces three-time MP Rajendra Agarwal who has been holding
the Meerut seat since 2004.
   Classification: pos
   Pos: 0.9874, Neg: 0.0126

Sentence 3: In a tweet on X, Govil thanked Prime Minister Narendra Modi for
entrusting him with such a "big responsibility" and said he will make efforts to
live up to the expectations of the people if he is elected.
   Classification: pos
   Pos: 0.9353, Neg: 0.0647

Sentence 4: "Heartfelt gratitude to Shri Narendra Modi ji and the selection
committee who have given me such a big responsibility by making me the MP
candidate of Meerut.
   Classification: pos
   Pos: 0.8767, Neg: 0.1233

Sentence 5: I will make every effort to fully live up to the trust of the BJP
and the expectations of the people.
   Classification: pos
   Pos: 0.7226, Neg: 0.2774

Sentence 6: Jai Shri Ram," he wrote in Hindi.
   Classification: neg
   Pos: 0.0087, Neg: 0.9913

Sentence 7: Govil was amongst the celebrities who had attended the grand
consecration ceremony of the Ram Temple in Ayodhya in January.
   Classification: pos
   Pos: 0.9157, Neg: 0.0843
```

```
Sentence 8: The veteran actor joined the BJP in 2021 and praised PM Modi for his
government's policies and changing the political narratives.
   Classification: pos
   Pos: 0.9405, Neg: 0.0595

Sentence 9: Notably, Govil had essayed the role of PM Modi in the latest
released film 'Article 370', which stars Yami Gautam.
   Classification: neg
   Pos: 0.4199, Neg: 0.5801

Sentence 10: Hailing from Meerut, Govil enrolled for a BSc before he became an
actor.
   Classification: neg
   Pos: 0.3206, Neg: 0.6794

Sentence 11: His debut film, Paheli, was released in 1977.
   Classification: pos
   Pos: 0.7786, Neg: 0.2214

Sentence 12: Govil became a household name after he played the role of Lord Ram
in Ramanand Sagar's TV adaption of Ramayan in the late 1980s.
   Classification: pos
   Pos: 0.6082, Neg: 0.3918

Sentence 13: He then went on to act in several television shows and also worked
in a number of Odia, Telugu, Bhojpuri and Braj Bhasha films.Published By:
Prateek ChakrabortyPublished On: Mar 24, 2024ALSO READ |  BJP drops Varun Gandhi
from Pilibhit, retains mother Maneka in SultanpurALSO READ |  BJP leader VK
Singh to not contest 2024 Lok Sabha pollsMust Watch
   Classification: pos
   Pos: 0.7879, Neg: 0.2121
```

0.0.6 10. (Spell CheckaProjectGutenbergBook)DownloadaProjectGutenbergbookandcreateaTextB Tokenize the TextBlob into Words and determine whether any are misspelled. If so, display the possible corrections.

[13]:
```
import requests
from textblob import TextBlob


url = "https://www.gutenberg.org/files/1342/1342-0.txt"
response = requests.get(url)
text = response.text
blob = TextBlob(text)
```

```python
# Tokenize and check spelling
for word in blob.words[:100]:
    if word.spellcheck()[0][1] < 1.0:
        print(f"Word: {word}, Suggestions: {word.spellcheck()}")
```

```
Word: START, Suggestions: [('START', 0.0)]
Word: OF, Suggestions: [('of', 0.17512732555086896), ('to',
0.12586415107548524), ('in', 0.0964786390605037), ('a', 0.09256261266779844),
('he', 0.05425993664350596), ('it', 0.046734165251938325), ('is',
0.04276563347743144), ('as', 0.0352836165706985), ('i', 0.03363407249243047),
('at', 0.029713670651241753), ('by', 0.029481771881617866), ('on',
0.02906610427568826), ('be', 0.026930885415755116), ('s', 0.024660027652834416),
('or', 0.023417400283529063), ('an', 0.01497716015891629), ('so',
0.01320072807462765), ('if', 0.010382939251273255), ('no',
0.010273553039186516), ('up', 0.009993524336244465), ('my',
0.009840383639323031), ('me', 0.008400861088261546), ('we',
0.008339604809492973), ('do', 0.006576299070654742), ('t',
0.005766841101212874), ('go', 0.003959780877539948), ('am',
0.0032640845686682887), ('us', 0.002992806762693176), ('ll',
0.0018683165024415002), ('oh', 0.0017939338782225179), ('mr',
0.00157516145404904), ('pp', 0.001347638132908623), ('o',
0.0011244902602516758), ('de', 0.0010369812905822848), ('ah',
0.0009713495633302413), ('ff', 0.0008313352118592156), ('re',
0.000826959763375746), ('d', 0.00078758072702452), ('m', 0.0007438262421898245),
('ve', 0.000669443617970842), ('f', 0.0006650681694873725), ('st',
0.0006431909270700247), ('c', 0.0006256891331361464), ('x',
0.0005950609937518596), ('e', 0.0005950609937518596), ('tm',
0.0005600574058841031), ('j', 0.0004025412604791991), ('eh',
0.0003894491502879046), ('b', 0.0003806640180618513 4), ('h',
0.0003456604301940949), ('ii', 0.00033690953322715577), ('ha',
0.0003281586362602 1666), ('w', 0.0003062813938428689), ('l',
0.0002975304968759298), ('p', 0.00028877959990899067), ('n',
0.0002712778059751 1244), ('g', 0.0002450251150742951), ('le',
0.00024064966659082557), ('la', 0.00024064966659082557), ('iv',
0.00024064966659082557), ('r', 0.00023189876962388645), ('v',
0.00022314787265694734), ('th', 0.00022314787265694734), ('dr',
0.00021002152720653867), ('y', 0.00017064249085531268), ('vi',
0.00016189159388837356), ('k', 0.00014438979995449533), ('ma',
0.00013563890298755622), ('co', 0.00013126345450408666), ('xi',
0.00012251255753714755), ('ix', 0.00012251255753714755), ('u',
0.00010938621208673889), ('en', 0.00010938621208673889), ('xv',
0.00010501076360326933), ('un', 8.313352118592156e-05), ('et',
8.313352118592156e-05), ('il', 7.8758072702452e-05), ('ex',
7.8758072702452e-05), ('au', 7.8758072702452e-05), ('je',
7.438262421898245e-05), ('pa', 6.563172725204333e-05), ('xx',
6.125627876857377e-05), ('ed', 6.125627876857377e-05), ('mb',
5.6880830285104224e-05), ('ne', 5.2505381801634667e-05), ('du',
```

5.2505381801634667e-05), ('cf', 4.812993331816511e-05), ('ce',
4.812993331816511e-05), ('ax', 4.812993331816511e-05), ('rd',
4.375448483469556e-05), ('hm', 4.375448483469556e-05), ('ze',
3.9379036351226e-05), ('vs', 3.9379036351226e-05), ('oo', 3.9379036351226e-05),
('ho', 3.9379036351226e-05), ('z', 3.5003587867756444e-05), ('ye',
3.5003587867756444e-05), ('qu', 3.5003587867756444e-05), ('op',
3.5003587867756444e-05), ('fe', 3.5003587867756444e-05), ('va',
3.062813938428689e-05), ('ti', 3.062813938428689e-05), ('ku',
3.062813938428689e-05), ('ta', 2.6252690900817333e-05), ('pg',
2.6252690900817333e-05), ('nd', 2.6252690900817333e-05), ('si',
2.187724241734778e-05), ('q', 2.187724241734778e-05), ('fr',
2.187724241734778e-05), ('er', 2.187724241734778e-05), ('em',
2.187724241734778e-05), ('ut', 1.7501793933878222e-05), ('tu',
1.7501793933878222e-05), ('sa', 1.7501793933878222e-05), ('gr',
1.7501793933878222e-05), ('fo', 1.7501793933878222e-05), ('da',
1.7501793933878222e-05), ('se', 1.3126345450408667e-05), ('po',
1.3126345450408667e-05), ('ou', 1.3126345450408667e-05), ('os',
1.3126345450408667e-05), ('na', 1.3126345450408667e-05), ('mi',
1.3126345450408667e-05), ('es', 1.3126345450408667e-05), ('ch',
1.3126345450408667e-05), ('ay', 1.3126345450408667e-05), ('ak',
1.3126345450408667e-05), ('ai', 1.3126345450408667e-05), ('zu',
8.750896966939111e-06), ('wm', 8.750896966939111e-06), ('pe',
8.750896966939111e-06), ('ox', 8.750896966939111e-06), ('oe',
8.750896966939111e-06), ('ni', 8.750896966939111e-06), ('ms',
8.750896966939111e-06), ('ke', 8.750896966939111e-06), ('jr',
8.750896966939111e-06), ('hi', 8.750896966939111e-06), ('ft',
8.750896966939111e-06), ('di', 8.750896966939111e-06), ('cm',
8.750896966939111e-06), ('ba', 8.750896966939111e-06), ('ab',
8.750896966939111e-06), ('wo', 4.3754484834695555e-06), ('wh',
4.3754484834695555e-06), ('wa', 4.3754484834695555e-06), ('vy',
4.3754484834695555e-06), ('vo', 4.3754484834695555e-06), ('tz',
4.3754484834695555e-06), ('ty', 4.3754484834695555e-06), ('tt',
4.3754484834695555e-06), ('tr', 4.3754484834695555e-06), ('te',
4.3754484834695555e-06), ('sq', 4.3754484834695555e-06), ('sn',
4.3754484834695555e-06), ('sg', 4.3754484834695555e-06), ('sd',
4.3754484834695555e-06), ('ri', 4.3754484834695555e-06), ('ra',
4.3754484834695555e-06), ('pm', 4.3754484834695555e-06), ('oz',
4.3754484834695555e-06), ('om', 4.3754484834695555e-06), ('ok',
4.3754484834695555e-06), ('ny', 4.3754484834695555e-06), ('nm',
4.3754484834695555e-06), ('nl', 4.3754484834695555e-06), ('mt',
4.3754484834695555e-06), ('mo', 4.3754484834695555e-06), ('mm',
4.3754484834695555e-06), ('md', 4.3754484834695555e-06), ('ly',
4.3754484834695555e-06), ('lo', 4.3754484834695555e-06), ('li',
4.3754484834695555e-06), ('lb', 4.3754484834695555e-06), ('km',
4.3754484834695555e-06), ('ka', 4.3754484834695555e-06), ('ja',
4.3754484834695555e-06), ('id', 4.3754484834695555e-06), ('ga',
4.3754484834695555e-06), ('fn', 4.3754484834695555e-06), ('ev',
4.3754484834695555e-06), ('el', 4.3754484834695555e-06), ('eg',

```
4.3754484834695555e-06), ('dy', 4.3754484834695555e-06), ('cy',
4.3754484834695555e-06), ('ca', 4.3754484834695555e-06), ('bu',
4.3754484834695555e-06), ('br', 4.3754484834695555e-06), ('bl',
4.3754484834695555e-06), ('bc', 4.3754484834695555e-06), ('ar',
4.3754484834695555e-06), ('al', 4.3754484834695555e-06), ('ad',
4.3754484834695555e-06), ('ac', 4.3754484834695555e-06)]
Word: THE, Suggestions: [('THE', 0.0)]
Word: PROJECT, Suggestions: [('PROJECT', 0.0)]
Word: GUTENBERG, Suggestions: [('GUTENBERG', 0.0)]
Word: EBOOK, Suggestions: [('EBOOK', 0.0)]
Word: GEORGE, Suggestions: [('GEORGE', 0.0)]
Word: ALLEN, Suggestions: [('ALLEN', 0.0)]
Word: PUBLISHER, Suggestions: [('PUBLISHER', 0.0)]
Word: CHARING, Suggestions: [('CHARING', 0.0)]
Word: CROSS, Suggestions: [('CROSS', 0.0)]
Word: ROAD, Suggestions: [('ROAD', 0.0)]
Word: LONDON, Suggestions: [('LONDON', 0.0)]
Word: RUSKIN, Suggestions: [('RUSKIN', 0.0)]
Word: HOUSE, Suggestions: [('HOUSE', 0.0)]
Word: Reading, Suggestions: [('Leading', 0.4791666666666667), ('Reading',
0.475), ('Heading', 0.041666666666666664), ('Beading', 0.004166666666666667)]
Word: Jane, Suggestions: [('Lane', 0.5625), ('Mane', 0.1875), ('Cane',
0.10416666666666667), ('Pane', 0.041666666666666664), ('Jane',
0.041666666666666664), ('Sane', 0.020833333333333332), ('Dane',
0.020833333333333332), ('Bane', 0.020833333333333332)]
Word: Letters, Suggestions: [('Letters', 0.9908256880733946), ('Fetters',
0.009174311926605505)]
Word: PRIDE, Suggestions: [('PRIDE', 0.0)]
Word: PREJUDICE, Suggestions: [('PREJUDICE', 0.0)]
Word: Jane, Suggestions: [('Lane', 0.5625), ('Mane', 0.1875), ('Cane',
0.10416666666666667), ('Pane', 0.041666666666666664), ('Jane',
0.041666666666666664), ('Sane', 0.020833333333333332), ('Dane',
0.020833333333333332), ('Bane', 0.020833333333333332)]
Word: Austen, Suggestions: [('Listen', 0.704225352112676), ('Mustn',
0.09859154929577464), ('Hasten', 0.07042253521126761), ('Fasten',
0.028169014084507043), ('Dusted', 0.028169014084507043), ('Muster',
0.02112676056338028), ('Russen', 0.014084507042253521), ('Austin',
0.014084507042253521), ('Rustan', 0.007042253521126761), ('Luster',
0.007042253521126761), ('Custer', 0.007042253521126761)]
Word: Saintsbury, Suggestions: [('Saintsbury', 0.0)]
Word: Hugh, Suggestions: [('Hugh', 0.7), ('Ugh', 0.3)]
Word: Ruskin, Suggestions: [('Skin', 0.8746223564954683), ('Tushin',
0.11329305135951662), ('Muslin', 0.0075528700906344415), ('Austin',
0.0030211480362537764), ('Pushkin', 0.0015105740181268882)]
Word: Charing, Suggestions: [('Sharing', 0.92), ('Charing', 0.08)]
Word: House, Suggestions: [('House', 0.977810650887574), ('Rouse',
0.011834319526627219), ('Mouse', 0.008875739644970414), ('Louse',
0.0014792899408284023)]
```

```
Word: Cross, Suggestions: [('Cross', 0.6956521739130435), ('Gross',
0.2028985507246377), ('Ross', 0.10144927536231885)]
Word: Road, Suggestions: [('Road', 0.9701492537313433), ('Load',
0.029850746268656716)]
Word: Allen, Suggestions: [('Ellen', 0.5), ('Allen', 0.5)]
Word: CHISWICK, Suggestions: [('CHISWICK', 0.0)]
Word: PRESS, Suggestions: [('PRESS', 0.0)]
Word: CHARLES, Suggestions: [('CHARLES', 0.0)]
Word: WHITTINGHAM, Suggestions: [('WHITTINGHAM', 0.0)]
Word: AND, Suggestions: [('AND', 0.0)]
Word: CO, Suggestions: [('of', 0.17512732555086896), ('to',
0.12586415107548524), ('in', 0.0964786390605037), ('a', 0.09256261266779844),
('he', 0.05425993664350596), ('it', 0.046734165251938325), ('is',
0.04276563347743144), ('as', 0.0352836165706985), ('i', 0.03363407249243047),
('at', 0.029713670651241753), ('by', 0.029481771881617866), ('on',
0.02906610427568826), ('be', 0.026930885415755116), ('s', 0.024660027652834416),
('or', 0.023417400283529063), ('an', 0.01497716015891629), ('so',
0.01320072807462765), ('if', 0.010382939251273255), ('no',
0.010273553039186516), ('up', 0.009993524336244465), ('my',
0.009840383639323031), ('me', 0.008400861088261546), ('we',
0.008339604809492973), ('do', 0.006576299070654742), ('t',
0.005766841101212874), ('go', 0.003959780877539948), ('am',
0.0032640845686682887), ('us', 0.002992806762693176), ('ll',
0.0018683165024415002), ('oh', 0.0017939338782225179), ('mr',
0.00157516145404904), ('pp', 0.001347638132908623), ('o',
0.0011244902602516758), ('de', 0.0010369812905822848), ('ah',
0.0009713495633302413), ('ff', 0.0008313352118592156), ('re',
0.000826959763375746), ('d', 0.00078758072702452), ('m', 0.0007438262421898245),
('ve', 0.000669443617970842), ('f', 0.0006650681694873725), ('st',
0.0006431909270700247), ('c', 0.0006256891331361464), ('x',
0.0005950609937518596), ('e', 0.0005950609937518596), ('tm',
0.0005600574058841031), ('j', 0.0004025412604791991), ('eh',
0.00038941491502879046), ('b', 0.00038066401806185134), ('h',
0.0003456604301940949), ('ii', 0.00033690953322715577), ('ha',
0.00032815863626021666), ('w', 0.0003062813938428689), ('l',
0.00029753049687592985), ('p', 0.00028877959990899067), ('n',
0.00027127780597511244), ('g', 0.0002450251150742951), ('le',
0.00024064966659082557), ('la', 0.00024064966659082557), ('iv',
0.00024064966659082557), ('r', 0.00023189876962388645), ('v',
0.00022314787265694734), ('th', 0.00022314787265694734), ('dr',
0.00021002152720653867), ('y', 0.00017064249085531268), ('vi',
0.00016189159388837356), ('k', 0.00014438979995449533), ('ma',
0.00013563890298755622), ('co', 0.00013126345450408666), ('xi',
0.00012251255753714755), ('ix', 0.00012251255753714755), ('u',
0.00010938621208673889), ('en', 0.00010938621208673889), ('xv',
0.00010501076360326933), ('un', 8.313352118592156e-05), ('et',
8.313352118592156e-05), ('il', 7.8758072702452e-05), ('ex',
7.8758072702452e-05), ('au', 7.8758072702452e-05), ('je',
```

7.438262421898245e-05), ('pa', 6.563172725204333e-05), ('xx',
6.125627876857377e-05), ('ed', 6.125627876857377e-05), ('mb',
5.6880830285104224e-05), ('ne', 5.2505381801634667e-05), ('du',
5.2505381801634667e-05), ('cf', 4.812993331816511e-05), ('ce',
4.812993331816511e-05), ('ax', 4.812993331816511e-05), ('rd',
4.375448483469556e-05), ('hm', 4.375448483469556e-05), ('ze',
3.9379036351226e-05), ('vs', 3.9379036351226e-05), ('oo', 3.9379036351226e-05),
('ho', 3.9379036351226e-05), ('z', 3.5003587867756444e-05), ('ye',
3.5003587867756444e-05), ('qu', 3.5003587867756444e-05), ('op',
3.5003587867756444e-05), ('fe', 3.5003587867756444e-05), ('va',
3.062813938428689e-05), ('ti', 3.062813938428689e-05), ('ku',
3.062813938428689e-05), ('ta', 2.6252690900817333e-05), ('pg',
2.6252690900817333e-05), ('nd', 2.6252690900817333e-05), ('si',
2.187724241734778e-05), ('q', 2.187724241734778e-05), ('fr',
2.187724241734778e-05), ('er', 2.187724241734778e-05), ('em',
2.187724241734778e-05), ('ut', 1.7501793933878222e-05), ('tu',
1.7501793933878222e-05), ('sa', 1.7501793933878222e-05), ('gr',
1.7501793933878222e-05), ('fo', 1.7501793933878222e-05), ('da',
1.7501793933878222e-05), ('se', 1.3126345450408667e-05), ('po',
1.3126345450408667e-05), ('ou', 1.3126345450408667e-05), ('os',
1.3126345450408667e-05), ('na', 1.3126345450408667e-05), ('mi',
1.3126345450408667e-05), ('es', 1.3126345450408667e-05), ('ch',
1.3126345450408667e-05), ('ay', 1.3126345450408667e-05), ('ak',
1.3126345450408667e-05), ('ai', 1.3126345450408667e-05), ('zu',
8.750896966939111e-06), ('wm', 8.750896966939111e-06), ('pe',
8.750896966939111e-06), ('ox', 8.750896966939111e-06), ('oe',
8.750896966939111e-06), ('ni', 8.750896966939111e-06), ('ms',
8.750896966939111e-06), ('ke', 8.750896966939111e-06), ('jr',
8.750896966939111e-06), ('hi', 8.750896966939111e-06), ('ft',
8.750896966939111e-06), ('di', 8.750896966939111e-06), ('cm',
8.750896966939111e-06), ('ba', 8.750896966939111e-06), ('ab',
8.750896966939111e-06), ('wo', 4.3754484834695555e-06), ('wh',
4.3754484834695555e-06), ('wa', 4.3754484834695555e-06), ('vy',
4.3754484834695555e-06), ('vo', 4.3754484834695555e-06), ('tz',
4.3754484834695555e-06), ('ty', 4.3754484834695555e-06), ('tt',
4.3754484834695555e-06), ('tr', 4.3754484834695555e-06), ('te',
4.3754484834695555e-06), ('sq', 4.3754484834695555e-06), ('sn',
4.3754484834695555e-06), ('sg', 4.3754484834695555e-06), ('sd',
4.3754484834695555e-06), ('ri', 4.3754484834695555e-06), ('ra',
4.3754484834695555e-06), ('pm', 4.3754484834695555e-06), ('oz',
4.3754484834695555e-06), ('om', 4.3754484834695555e-06), ('ok',
4.3754484834695555e-06), ('ny', 4.3754484834695555e-06), ('nm',
4.3754484834695555e-06), ('nl', 4.3754484834695555e-06), ('mt',
4.3754484834695555e-06), ('mo', 4.3754484834695555e-06), ('mm',
4.3754484834695555e-06), ('md', 4.3754484834695555e-06), ('ly',
4.3754484834695555e-06), ('lo', 4.3754484834695555e-06), ('li',
4.3754484834695555e-06), ('lb', 4.3754484834695555e-06), ('km',
4.3754484834695555e-06), ('ka', 4.3754484834695555e-06), ('ja',

```
4.3754484834695555e-06), ('id', 4.3754484834695555e-06), ('ga',
4.3754484834695555e-06), ('fn', 4.3754484834695555e-06), ('ev',
4.3754484834695555e-06), ('el', 4.3754484834695555e-06), ('eg',
4.3754484834695555e-06), ('dy', 4.3754484834695555e-06), ('cy',
4.3754484834695555e-06), ('ca', 4.3754484834695555e-06), ('bu',
4.3754484834695555e-06), ('br', 4.3754484834695555e-06), ('bl',
4.3754484834695555e-06), ('bc', 4.3754484834695555e-06), ('ar',
4.3754484834695555e-06), ('al', 4.3754484834695555e-06), ('ad',
4.3754484834695555e-06), ('ac', 4.3754484834695555e-06)]
Word: TOOKS, Suggestions: [('TOOKS', 0.0)]
Word: COURT, Suggestions: [('COURT', 0.0)]
Word: CHANCERY, Suggestions: [('CHANCERY', 0.0)]
Word: LANE, Suggestions: [('LANE', 0.0)]
Word: LONDON, Suggestions: [('LONDON', 0.0)]
Word: To, Suggestions: [('To', 0.7805182471849138), ('So', 0.0818613485280152),
('No', 0.06370913037579705), ('Do', 0.04078144078144078), ('Go',
0.02455569122235789), ('O', 0.006973273639940307), ('Co', 0.000814000814000814),
('Oo', 0.0002442002442002442), ('Ho', 0.0002442002442002442), ('Fo',
0.0001085334418667752), ('Po', 8.14000814000814e-05), ('Wo',
2.71333604666938e-05), ('Vo', 2.71333604666938e-05), ('Mo',
2.71333604666938e-05), ('Lo', 2.71333604666938e-05)]
Word: Comyns, Suggestions: [('Romans', 0.75), ('Omens', 0.25)]
Word: Hugh, Suggestions: [('Hugh', 0.7), ('Ugh', 0.3)]
Word: PREFACE, Suggestions: [('PREFACE', 0.0)]
Word: Walt, Suggestions: [('Salt', 0.6071428571428571), ('Halt',
0.35714285714285715), ('Walt', 0.017857142857142856), ('Malt',
0.017857142857142856)]
```

0.0.7 11.

- **Write a Python program that takes user input in English and translates it to French, Spanish, and German using TextBlob.**
- **Create a program that takes multiple user-inputted sentences, analyzes polarity and subjectivity, and categorizes them as objective/subjective and positive/negative/neutral.**
- **Develop a function that takes a paragraph, splits it into sentences, and calculates the sentiment score for each sentence individually.**
- **Write a program that takes a sentence as input and prints each word along with its POS tag using TextBlob.**
- **Create a function that takes a user-inputted word, checks its spelling using TextBlob, and suggests top 3 closest words if a mistake is found.**
- **Build a Python script that extracts all adjectives from a given paragraph and prints them in order of occurrence.**
- **Write a program that takes a news article as input and extracts the top 5 most common noun phrases as keywords.**

- **Write a program that takes a news article as input and extracts the top 5 most common noun phrases as keywords.**
- **Write a program that summarizes a given paragraph by keeping only the most informative sentences, based on noun phrase frequency.**

[16]:
```
#Write a Python program that takes user input in English and translates it to␣
 ↪French, Spanish, and German using TextBlob
from deep_translator import GoogleTranslator

text = input("Enter a sentence in English: ")

print("French:", GoogleTranslator(source='auto', target='fr').translate(text))
print("Spanish:", GoogleTranslator(source='auto', target='es').translate(text))
print("German:", GoogleTranslator(source='auto', target='de').translate(text))
```

```
Enter a sentence in English:  Hello, how are you?

French: Bonjour comment allez-vous?
Spanish: ¿Hola, cómo estás?
German: Hallo, wie geht es dir?
```

[17]:
```
#Create a program that takes multiple user-inputted sentences, analyzes␣
 ↪polarity and subjectivity, and categorizes them as objective/subjective and␣
 ↪positive/negative/neutral.
from textblob import TextBlob

sentences = input("Enter multiple sentences: ").split(".")
for sentence in sentences:
    blob = TextBlob(sentence.strip())
    polarity = blob.sentiment.polarity
    subjectivity = blob.sentiment.subjectivity

    if not sentence.strip():
        continue

    print(f"\nSentence: {sentence.strip()}")
    print("Subjectivity:", "Subjective" if subjectivity > 0.5 else "Objective")
    if polarity > 0:
        print("Polarity: Positive")
    elif polarity < 0:
        print("Polarity: Negative")
    else:
        print("Polarity: Neutral")
```

```
Enter multiple sentences:  It is a fine day today. The weather is nice. But it
might not be a fine day tomorrow.
```

```
Sentence: It is a fine day today
Subjectivity: Objective
Polarity: Positive

Sentence: The weather is nice
Subjectivity: Subjective
Polarity: Positive

Sentence: But it might not be a fine day tomorrow
Subjectivity: Objective
Polarity: Positive
```

[18]:
```python
#Develop a function that takes a paragraph, splits it into sentences, and␣
 ↪calculates the sentiment score for each sentence individually.

from textblob import TextBlob

paragraph = input("Enter a paragraph: ")
blob = TextBlob(paragraph)

for i, sentence in enumerate(blob.sentences, 1):
    print(f"Sentence {i}: {sentence}")
    print("Polarity:", sentence.sentiment.polarity)
    print("Subjectivity:", sentence.sentiment.subjectivity)
    print()
```

```
Enter a paragraph:  After a long day at work, she walked into her apartment and
was greeted by the soft purring of her cat. The warm light in the living room
and the scent of her favorite candle made her feel instantly at peace. It was in
these quiet moments that she truly appreciated the little things in life

Sentence 1: After a long day at work, she walked into her apartment and was
greeted by the soft purring of her cat.
Polarity: 0.025
Subjectivity: 0.375

Sentence 2: The warm light in the living room and the scent of her favorite
candle made her feel instantly at peace.
Polarity: 0.375
Subjectivity: 0.7416666666666666

Sentence 3: It was in these quiet moments that she truly appreciated the little
things in life
Polarity: 0.00416666666666667
Subjectivity: 0.3111111111111111
```

[21]:
```
# Write a program that takes a sentence as input and prints each word along␣
 ↪with its POS tag using TextBlob.
from textblob import TextBlob

text = input("Enter a sentence: ")
blob = TextBlob(text)

for word, tag in blob.tags:
    print(f"{word}: {tag}")
```

```
Enter a sentence:  Hello, How are you?
```

```
Hello: NNP
How: NNP
are: VBP
you: PRP
```

[25]:
```
#Create a function that takes a user-inputted word, checks its spelling using␣
 ↪TextBlob, and suggests top 3 closest words if a mistake is found.
from textblob import Word

word = input("Enter a word: ")
w = Word(word)
corrected = w.correct()

print("Suggested correction:", corrected)
print("Top 3 guesses:", w.spellcheck()[:3])
```

```
Enter a word:  chini
```

```
Suggested correction: china
Top 3 guesses: [('china', 0.5657894736842105), ('chin', 0.39473684210526316),
('chink', 0.02631578947368421)]
```

[26]:
```
# Build a Python script that extracts all adjectives from a given paragraph and␣
 ↪prints them in order of occurrence.
from textblob import TextBlob

text = input("Enter a paragraph: ")
blob = TextBlob(text)

adjectives = [word for word, tag in blob.tags if tag == "JJ"]
print("Adjectives in order of occurrence:")
print(adjectives)
```

```
Enter a paragraph:  After a long day at work, she walked into her apartment and
was greeted by the soft purring of her cat. The warm light in the living room
and the scent of her favorite candle made her feel instantly at peace. It was in
these quiet moments that she truly appreciated the little things in life
```

```
Adjectives in order of occurrence:
['long', 'soft', 'warm', 'favorite', 'quiet', 'little']
```

[27]:
```python
#Write a program that takes a news article as input and extracts the top 5 most␣
 ↪common noun phrases as keywords.
from textblob import TextBlob
from collections import Counter

text = input("Paste the news article: ")
blob = TextBlob(text)

noun_phrases = Counter(blob.noun_phrases)
top_5 = noun_phrases.most_common(5)

print("Top 5 Keywords (Noun Phrases):")
for phrase, freq in top_5:
    print(f"{phrase} ({freq} times)")
```

```
Paste the news article:  After a long day at work, she walked into her apartment
and was greeted by the soft purring of her cat. The warm light in the living
room and the scent of her favorite candle made her feel instantly at peace. It
was in these quiet moments that she truly appreciated the little things in life

Top 5 Keywords (Noun Phrases):
long day (1 times)
warm light (1 times)
quiet moments (1 times)
```

[28]:
```python
# Write a program that summarizes a given paragraph by keeping only the most␣
 ↪informative sentences, based on noun phrase frequency.
from textblob import TextBlob
from collections import Counter

def summarize(text):
    blob = TextBlob(text)
    noun_phrases = Counter(blob.noun_phrases)

    scored_sentences = []
    for sentence in blob.sentences:
        score = sum(noun_phrases[np] for np in sentence.noun_phrases)
        scored_sentences.append((score, str(sentence)))

    # Sort sentences by score and take top 3
    top_sentences = sorted(scored_sentences, reverse=True)[:3]
    summary = " ".join(sentence for score, sentence in top_sentences)
    return summary

paragraph = input("Enter a paragraph to summarize: ")
```

```
print("\nSummary:")
print(summarize(paragraph))
```

```
Enter a paragraph to summarize:  After a long day at work, she walked into her
apartment and was greeted by the soft purring of her cat. The warm light in the
living room and the scent of her favorite candle made her feel instantly at
peace. It was in these quiet moments that she truly appreciated the little
things in life


Summary:
The warm light in the living room and the scent of her favorite candle made her
feel instantly at peace. It was in these quiet moments that she truly
appreciated the little things in life After a long day at work, she walked into
her apartment and was greeted by the soft purring of her cat.
```

0.0.8 12. Write a Python program that takes a word as input and returns:

- Its definition
- Its synonyms
- Its antonyms(if available)

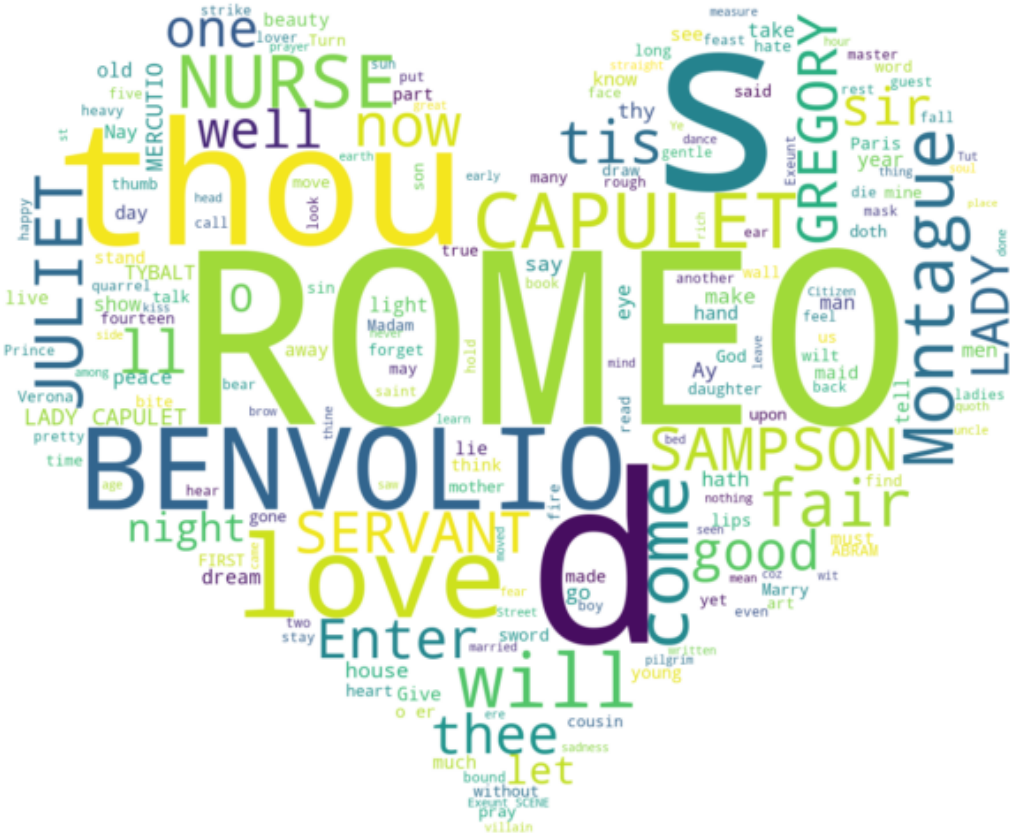
[30]:
```
from textblob import Word

word = input("Enter a word: ")
w = Word(word)
print("Definition:", w.definitions[0] if w.definitions else "Not found")
print("Synonyms:", w.synsets[0].lemma_names() if w.synsets else "Not found")
print("Antonyms:", w.synsets[0].lemmas()[0].antonyms()[0].name() if w.synsets␣
 ↪and w.synsets[0].lemmas()[0].antonyms() else "Not found")
```

```
Enter a word:  jolly

Definition: a happy party
Synonyms: ['jolly']
Antonyms: Not found
```

13. * Write a Python program that reads a .txt file, processes the text, and generates a word cloud visualization. * Create a word cloud in the shape of an object (e.g., a heart, star) using WordCloud and a mask image.

[34]:
```
from pathlib import Path
import imageio.v2 as imageio
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Read Text file
text = Path(r'romeoandjuliet.txt').read_text('utf-8')
```

```
# Read mask image
mask_image = imageio.imread('Heart.png')

wordcloud = WordCloud(colormap='viridis', mask=mask_image,␣
 ↪background_color='white')

wordcloud = wordcloud.generate(text)

plt.figure(figsize=(8,8))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```



0.0.9 14. (Textatistic: Readability of News Articles) Using the above techniques, download from several news sites current news articles on the same topic. Perform readability assessments on them to determine which sites are the most readable. For each article, calculate the average number of words per sentence, the average number of characters per word and the average number of syllables per word.

[1]:
```
import requests
from bs4 import BeautifulSoup
from textblob import TextBlob
import textstat

urls = [
    "https://www.bbc.com/news/world-us-canada-68787829",
    "https://edition.cnn.com/2024/04/06/politics/trump-trial-updates/index.
 ↪html",
    "https://www.reuters.com/world/us/us-employment-report-april-2024-04-05/",
]
for url in urls:
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    text = soup.get_text()
    blob = TextBlob(text)
    print(f"URL: {url}")
    print(f"Avg Words/Sentence: {len(blob.words) / len(blob.sentences)}")
    print(f"Avg Chars/Word: {sum(len(w) for w in blob.words) / len(blob.
 ↪words)}")
    print(f"Avg Syllables/Word: {textstat.syllable_count(text) / len(blob.
 ↪words)}")
```

```
URL: https://www.bbc.com/news/world-us-canada-68787829
Avg Words/Sentence: 16.75
Avg Chars/Word: 5.373134328358209
Avg Syllables/Word: 1.9850746268656716
URL: https://edition.cnn.com/2024/04/06/politics/trump-trial-updates/index.html
Avg Words/Sentence: 72.0
Avg Chars/Word: 15.76388888888889
Avg Syllables/Word: 3.9166666666666665
URL: https://www.reuters.com/world/us/us-employment-report-april-2024-04-05/
Avg Words/Sentence: 8.0
Avg Chars/Word: 5.875
Avg Syllables/Word: 2.0
```

0.0.10 17. (textblob.utils Utility Functions) Use strip punc and lowerstrip functions of TextBlob's textblob.utils module with all=True keyword argument to remove punctuation and to get a string in all lowercase letters with whitespace and punctuation removed. Experiment with each function on Romeo and Juliet

[4]:
```
from textblob.utils import strip_punc, lowerstrip

# Load the Romeo and Juliet text
with open("romeoandjuliet.txt", "r", encoding="utf-8") as file:
    text = file.read()


no_punctuation = strip_punc(text, all=True)
print("=== Text without punctuation ===")
print(no_punctuation[:500])


lower_cleaned = lowerstrip(text, all=True)
print("\n=== Lowercase + no punctuation ===")
print(lower_cleaned[:500])
```

```
=== Text without punctuation ===
ACT I
SCENE I A public place
Enter Sampson and Gregory armed with swords and bucklers

SAMPSON
Gregory on my word we'll not carry coals

GREGORY
No for then we should be colliers

SAMPSON
I mean if we be in choler we'll draw

GREGORY
Ay while you live draw your neck out o' the collar

SAMPSON
I strike quickly being moved

GREGORY
But thou art not quickly moved to strike

SAMPSON
A dog of the house of Montague moves me

GREGORY
```

```
To move is to stir and to be valiant is to stand therefore if thou ar

=== Lowercase + no punctuation ===
act i
scene i a public place
enter sampson and gregory armed with swords and bucklers

sampson
gregory on my word we'll not carry coals

gregory
no for then we should be colliers

sampson
i mean if we be in choler we'll draw

gregory
ay while you live draw your neck out o' the collar

sampson
i strike quickly being moved

gregory
but thou art not quickly moved to strike

sampson
a dog of the house of montague moves me

gregory
to move is to stir and to be valiant is to stand therefore if thou ar
```