Head:

python

CopyEdit

```
!pip install nltk

import nltk

nltk.data.path.append('Desktop')

from nltk.tokenize import sent_tokenize

nltk.download('punkt')
```

Tail:

python

CopyEdit

```
nltk.download('punkt')
```

Explanation:

This segment installs and imports the necessary NLTK (Natural Language Toolkit) package for text processing. The `sent_tokenize` function is used to split the text into sentences, and `nltk.download('punkt')` is used to download the tokenizers used for splitting sentences and words.

Head:

python

CopyEdit

```
text = """Hello Mr.smith,how are you doing today? The
weather is great, and city is awesome. The sky is
pinkish-blue. You shouldn't eat cardboard"""
```

```
tokenized_text = sent_tokenize(text)

print(tokenized_text)
```

Tail:

python

CopyEdit

```
print(tokenized_text)
```

Explanation:

Here, a sample text is defined, and `sent_tokenize` is used to split this text into sentences. The output is a list of sentences, which is printed for verification.

Head:

python

CopyEdit

```
from nltk.tokenize import word_tokenize

tokenized_word = word_tokenize(text)

print(tokenized_word)
```

Tail:

python

CopyEdit

```
print(tokenized_word)
```

Explanation:

This part tokenizes the text into individual words using `word_tokenize`. It breaks down the text into smaller components like words and punctuation. The output is printed to display the word-level tokenization.

Head:

python

CopyEdit

```
from nltk.probability import FreqDist

fdist = FreqDist(tokenized_word)

print(fdist)

fdist.most_common(2)
```

Tail:

python

CopyEdit

```
fdist.most_common(2)
```

Explanation:

A frequency distribution (`FreqDist`) is created for the tokenized words to count the frequency of each word in the text. The `.most_common(2)` method is used to print the two most common words in the text.

Head:

python

CopyEdit

```
import matplotlib.pyplot as plt

fdist.plot(30, cumulative=False)

plt.show()
```

Tail:

python

CopyEdit

```
plt.show()
```

Explanation:

This segment uses matplotlib to plot the frequency distribution of words, showing the most frequent words in a histogram. The `cumulative=False` argument ensures that each frequency count is plotted individually rather than cumulatively.

Head:

python

CopyEdit

```
nltk.download('stopwords')

from nltk.corpus import stopwords

stop_words = set(stopwords.words("english"))

print(stop_words)
```

Tail:

python

CopyEdit

```
print(stop_words)
```

Explanation:

Here, the stopwords corpus is downloaded using `nltk.download('stopwords')`. A set of common English stopwords is then retrieved from the NLTK corpus. These words, such as "and", "the", "is", are typically removed during text preprocessing because they don't carry meaningful content.

Head:

python

CopyEdit

```
from nltk.tokenize import word_tokenize

text1 = """Hello Mr.smith,how are you doing today?"""

tokenized_sent = word_tokenize(text1)

print(tokenized_sent)
```

Tail:

python

CopyEdit

```
print(tokenized_sent)
```

Explanation:

A new sample sentence is tokenized using `word_tokenize` to demonstrate how the words are split from a sentence. The tokenized words are printed to the console.

Head:

python

CopyEdit

```
filtered_sent = []
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentences:", tokenized_sent)
print("Filtered Sentence:", filtered_sent)
```

Tail:

python

CopyEdit

```
print("Filtered Sentence:", filtered_sent)
```

Explanation:

The code loops over the tokenized words and removes any stopwords (e.g., "how", "are") from the list. The final list, `filtered_sent`, only includes meaningful words from the sentence.

Head:

python

CopyEdit

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

```
stemmed_words = []

for w in filtered_sent:

    stemmed_words.append(ps.stem(w))
```

Tail:

python

CopyEdit

```
print("Stemmed Sentence:", stemmed_words)
```

Explanation:

Here, a PorterStemmer is used to perform stemming, which reduces words to their root forms (e.g., "running" becomes "run"). The loop applies stemming to the filtered sentence, and the resulting list is printed.

Head:

python

CopyEdit

```
nltk.download('wordnet')

nltk.download('omw-1.4')

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()
```

Tail:

python

CopyEdit

```
nltk.download('omw-1.4')
```

Explanation:

The `WordNetLemmatizer` is imported to perform lemmatization. Unlike stemming, lemmatization converts a word to its base or dictionary form (e.g., "running" becomes "run"). The necessary corpora are downloaded for the lemmatizer.

Head:

python

CopyEdit

```
word = "flying"

print("Lemmatized word:", lem.lemmatize(word, "v"))
```

Tail:

python

CopyEdit

```
print("Stemmed word:", stem.stem(word))
```

Explanation:

The code demonstrates both lemmatization and stemming on the word "flying". The lemmatizer converts it to "fly" (verb form), and the stemmer reduces it to "fli". This shows the difference between these two text normalization techniques.

Head:

python

CopyEdit

```
sent = "Albert Einstein was born in Ulm, Germany in 1879."

tokens = nltk.word_tokenize(sent)

print(tokens)
```

Tail:

python

CopyEdit

```
print(tokens)
```

Explanation:

A new sentence is tokenized into words using `word_tokenize` and the tokenized words are printed.

Head:

python

CopyEdit

```
nltk.download('averaged_perceptron_tagger')

nltk.pos_tag(tokens)
```

Tail:

python

CopyEdit

```
nltk.pos_tag(tokens)
```

Explanation:

This part performs Part-of-Speech (POS) tagging on the tokenized words, identifying the grammatical role of each word (e.g., noun, verb, adjective). It uses the averaged perceptron tagger provided by NLTK.

Head:

python

CopyEdit

```
from sklearn.feature_extraction.text import
TfidfVectorizer

corpus = [

    "Sachin was the GOAT of the previous generation",

    "Virat is the GOAT of the this generation",

    "Shubman will be the GOAT of the next generation"

]

vectorizer = TfidfVectorizer()

matrix = vectorizer.fit(corpus)
```

Tail:

python

CopyEdit

```
matrix.vocabulary_

tfidf_matrix = vectorizer.transform(corpus)

print(tfidf_matrix)
```

```
print(vectorizer.get_feature_names_out())
```

Explanation:

A TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is used to convert the text corpus into numerical features. This technique calculates the importance of each word in the context of the document and corpus. The resulting matrix shows the TF-IDF values for each term across the documents. The vocabulary and feature names are printed for analysis.