In [1]:
```
import nltk
import pandas as pd
import numpy as np
from collections import Counter
from tabulate import tabulate
nltk.download('punkt')
```

```
C:\Users\pronn\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this vers
of SciPy (detected version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\pronn\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]: True

Word Frequency Count

In [2]:
```
text = input("Enter a text: ")

sentence = nltk.sent_tokenize(text)
tokens = [nltk.word_tokenize(sentence) for sentence in sentence]

print("Word  Frequency Count")
for sentence_tokens in tokens :
    word_counts = pd.value_counts(np.array(sentence_tokens))
    print(word_counts)

flat_tokens = [word.lower() for sentence_tokens in tokens for word in sentence_tokens]
unique_words = list(set(flat_tokens))

print (flat_tokens, unique_words)
```

```
Enter a text: nory was a catholic because her mother , and 's father his or had been
Word  Frequency Count
nory        1
was         1
a           1
catholic    1
because     1
her         1
mother      1
,           1
and         1
's          1
father      1
his         1
or          1
had         1
been        1
dtype: int64
['nory', 'was', 'a', 'catholic', 'because', 'her', 'mother', ',', 'and', "'s", 'father', 'his', 'or', 'had', 'been'] ['his', 'mother', 'or',
s', 'because', ',', "'s", 'nory', 'and', 'a', 'had', 'father', 'been', 'catholic', 'her']
```

Bigram Count Table

In [3]:
```
bigrams = list(nltk.bigrams(flat_tokens))
unigram_counter = Counter(flat_tokens)
matrix_size = len(unique_words)
matrix = [[0] * matrix_size for _ in range(matrix_size)]
word_to_index = {word : index for index, word in enumerate(unique_words)}

for bigram in bigrams :
    word1, word2 = bigram
    index1 = word_to_index[word1]
    index2 = word_to_index[word2]
    matrix[index1][index2] += 1

header = [''] + unique_words
matrix_with_headers = [[unique_words[i]] + matrix[i] for i in range(matrix_size)]

print("\nBigram Count Table:")
print(tabulate(matrix_with_headers, headers=header, tablefmt='grid'))
```

Bigram Count Table:

	his	mother	or	was	because	,	's	nory	and	a	had	father	been	catholic	her
his	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
mother	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
or	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
was	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
because	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
,	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
's	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
nory	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
and	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
had	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
father	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
been	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
catholic	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
her	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bigram Probability table

In [4]:
```
bigrams = list(nltk.bigrams(flat_tokens))
unigram_counter = Counter(flat_tokens)
matrix_size = len(unique_words)
matrix = [[0] * matrix_size for _ in range(matrix_size)]
word_to_index = {word : index for index, word in enumerate(unique_words)}

for bigram in bigrams :
    word1, word2 = bigram
    index1 = word_to_index[word1]
    index2 = word_to_index[word2]
    matrix[index1][index2] += 1 / unigram_counter[word1]

header = [''] + unique_words
matrix_with_headers = [[unique_words[i]] + matrix[i] for i in range(matrix_size)]

print("\nBigram Probability Table:")
print(tabulate(matrix_with_headers, headers=header, tablefmt='grid'))
```

Bigram Probability Table:

	his	mother	or	was	because	,	's	nory	and	a	had	father	been	catholic	her
his	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
mother	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
or	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
was	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
because	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
,	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
's	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
nory	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
and	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
had	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
father	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
been	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
catholic	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Predicting next word

In [5]:
```
def predict_next_word(start_word, matrix_with_headers, word_to_index):
    if start_word in word_to_index :
        start_index = word_to_index[start_word]
        next_word_probs = matrix_with_headers[start_index][1:]
        max_prob_index = next_word_probs.index(max(next_word_probs))
        return unique_words[max_prob_index]
    else:
        return "Starting word not found in the text."

starting_word = input("Enter a starting word: ")
predicted_next_word = predict_next_word(starting_word, matrix_with_headers, word_to_index)
print("Predicted next word:", predicted_next_word)
```

```
Enter a starting word: mother
Predicted next word: ,
```