CORPUS

```
D1 = "I enjoy reading books on history"
D2 = "The pizza was cold and tasteless"
D3 = "She received a promotion at work"
D4 = "Heavy rain caused flooding downtown"
D5 = "The movie was surprisingly good"
D6 = "My laptop battery dies quickly"
D7 = "The concert tickets sold out fast"
D8 = "He adopted a puppy from the shelter"
D9 = "The exam was more difficult than expected"
D10 = "Scientists discovered a new species of frog"
```

UNI-GRAM COUNTS

```
import collections

# Combine the text from the text documents
combined_text = f"{D1} {D2} {D3} {D4} {D5} {D6} {D7} {D8} {D9} {D10}"

# Tokenize the combined text into words and convert to lowercase
words = combined_text.lower().split()

# Calculate unigram counts
unigram_counts = collections.Counter(words)

# Print the unigram counts
print("Unigram Counts:")
for word, count in unigram_counts.most_common():
    print(f"{word}: {count}")
#Vocabulary size is length of unigrams
V=len(unigram_counts)
print("Vocabulary Size=",V)
```

```
Unigram Counts:
the: 5
was: 3
a: 3
i: 1
enjoy: 1
reading: 1
books: 1
on: 1
history: 1
pizza: 1
cold: 1
and: 1
tasteless: 1
she: 1
received: 1
promotion: 1
at: 1
work: 1
heavy: 1
rain: 1
caused: 1
flooding: 1
downtown: 1
movie: 1
surprisingly: 1
good: 1
my: 1
laptop: 1
battery: 1
dies: 1
quickly: 1
concert: 1
tickets: 1
sold: 1
out: 1
fast: 1
he: 1
adopted: 1
puppy: 1
from: 1
shelter: 1
```

```
exam: 1
more: 1
difficult: 1
than: 1
expected: 1
scientists: 1
discovered: 1
new: 1
species: 1
of: 1
frog: 1
Vocabulary Size= 52
```

BI-GRAM COUNTS

```
import collections
combined_text = f"{D1} {D2} {D3} {D4} {D5} {D6} {D7} {D8} {D9} {D10}"
words = combined_text.lower().split()
# Generate bigrams
bigrams = []
for i in range(len(words) - 1):
    bigrams.append((words[i], words[i+1]))
# Calculate bigram counts
bigram_counts = collections.Counter(bigrams)
# Print the bigram counts
print("\nBigram Counts:")
for bigram, count in bigram_counts.most_common():
    print(f"{bigram[0]} {bigram[1]}: {count}")
```

```
enjoy reading: 1
reading books: 1
books on: 1
on history: 1
history the: 1
the pizza: 1
pizza was: 1
was cold: 1
cold and: 1
and tasteless: 1
tasteless she: 1
she received: 1
received a: 1
a promotion: 1
promotion at: 1
at work: 1
work heavy: 1
heavy rain: 1
rain caused: 1
caused flooding: 1
flooding downtown: 1
downtown the: 1
the movie: 1
movie was: 1
was surprisingly: 1
surprisingly good: 1
good my: 1
my laptop: 1
laptop battery: 1
battery dies: 1
dies quickly: 1
quickly the: 1
the concert: 1
concert tickets: 1
tickets sold: 1
sold out: 1
out fast: 1
fast he: 1
he adopted: 1
adopted a: 1
a puppy: 1
puppy from: 1
from the: 1
the shelter: 1
shelter the: 1
the exam: 1
exam was: 1
was more: 1
more difficult: 1
```

```
scientists discovered: 1
discovered a: 1
a new: 1
new species: 1
species of: 1
of frog: 1
```

TRI-GRAM COUNTS

```
import collections
combined_text = f"{D1} {D2} {D3} {D4} {D5} {D6} {D7} {D8} {D9} {D10}"
words = combined_text.lower().split()
# Generate bigrams
Trigrams = []
for i in range(len(words) - 2):
    Trigrams.append((words[i], words[i+1], words[i+2]))
# Calculate bigram counts
Trigrams_counts = collections.Counter(Trigrams)
# Print the bigram counts
print("\nTrigrams Counts:")
for Trigrams, count in Trigrams_counts.most_common():
    print(f"{Trigrams[0]} {Trigrams[1]} {Trigrams[2]}: {count}")
```

```
Trigrams Counts:
i enjoy reading: 1
enjoy reading books: 1
reading books on: 1
books on history: 1
on history the: 1
history the pizza: 1
the pizza was: 1
pizza was cold: 1
was cold and: 1
cold and tasteless: 1
and tasteless she: 1
tasteless she received: 1
she received a: 1
received a promotion: 1
a promotion at: 1
promotion at work: 1
at work heavy: 1
work heavy rain: 1
heavy rain caused: 1
rain caused flooding: 1
caused flooding downtown: 1
flooding downtown the: 1
downtown the movie: 1
the movie was: 1
movie was surprisingly: 1
was surprisingly good: 1
surprisingly good my: 1
good my laptop: 1
my laptop battery: 1
laptop battery dies: 1
battery dies quickly: 1
dies quickly the: 1
quickly the concert: 1
the concert tickets: 1
concert tickets sold: 1
tickets sold out: 1
sold out fast: 1
out fast he: 1
fast he adopted: 1
he adopted a: 1
adopted a puppy: 1
a puppy from: 1
puppy from the: 1
from the shelter: 1
the shelter the: 1
shelter the exam: 1
the exam was: 1
exam was more: 1
was more difficult: 1
more difficult than: 1
difficult than expected: 1
than expected scientists: 1
expected scientists discovered: 1
scientists discovered a: 1
discovered a new: 1
a new species: 1
```

NEXT WORD PREDICTION USING BI-GRAM MODEL

```
def predict_next_word_bigram(word_sequence, bigram_counts, unigram_counts):
    # Tokenize the input sequence and get the last word
    words_in_sequence = word_sequence.lower().split()
    if not words_in_sequence:
        return "Please provide a word sequence."
    last_word = words_in_sequence[-1]

    # Find potential next words based on bigrams starting with last_word
    potential_next_words = {}
    for (w1, w2), count in bigram_counts.items():
        if w1 == last_word:
            potential_next_words[w2] = count

    if not potential_next_words:
        return f"No bigram found starting with '{last_word}'."

    # Calculate probabilities for potential next words
    # P(w2 | w1) = Count(w1, w2) / Count(w1)
    last_word_unigram_count = unigram_counts.get(last_word, 0)
    if last_word_unigram_count == 0:
        return f"'{last_word}' not found in unigram counts. Cannot predict next word."

    predicted_word = None
    max_probability = -1

    for next_word, bigram_count in potential_next_words.items():
        probability = bigram_count / last_word_unigram_count
        print("probability of  " f'{next_word}' " is ", probability)
        if probability > max_probability:
            max_probability = probability
            predicted_word = next_word

    return predicted_word

# Example usage:
# Ensure bigram_counts and unigram_counts are defined from previous cells
# (They are present in the kernel state.)

# Try with a word sequence
sequence1 = "a puppy"
next_word1 = predict_next_word_bigram(sequence1, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

sequence2 = "my laptop"
next_word2 = predict_next_word_bigram(sequence2, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")

sequence3 = "I am"
next_word3 = predict_next_word_bigram(sequence3, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence3}', predicted next word: '{next_word3}'")

sequence4 = "nonexistent word"
next_word4 = predict_next_word_bigram(sequence4, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence4}', predicted next word: '{next_word4}'")
```

```
probability of  from is  1.0
Given sequence: 'a puppy', predicted next word: 'from'
probability of  battery is  1.0
Given sequence: 'my laptop', predicted next word: 'battery'
Given sequence: 'I am', predicted next word: 'No bigram found starting with 'am'.'
Given sequence: 'nonexistent word', predicted next word: 'No bigram found starting with 'word'.'
```

DEPLOYMENT OF BI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_bigram(ip_text, bigram_counts, unigram_counts)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : My name
Given sequence: 'My name', predicted next word: 'No bigram found starting with 'name'.'
```

NEXT WORD PREDICTION USING TRI-GRAM MODEL

```
def predict_next_word_trigram(word_sequence, Trigrams_counts, bigram_counts):
    # Tokenize the input sequence
    words_in_sequence = word_sequence.lower().split()
    if not words_in_sequence:
        return "Please provide a word sequence."

    # Ensure at least two words for trigram prediction
    if len(words_in_sequence) < 2:
        return "Sequence must contain at least two words for trigram prediction."

    # Get the last two words as a tuple
    last_two_words_tuple = tuple(words_in_sequence[-2:])

    # Find potential next words based on trigrams starting with the last two words
    potential_next_words = {}
    for (w1, w2, w3), count in Trigrams_counts.items():
        if (w1, w2) == last_two_words_tuple:
            potential_next_words[w3] = count

    if not potential_next_words:
        return f"No trigram found starting with '{' '.join(last_two_words_tuple)}'."

    # Calculate probabilities for potential next words
    # P(w3 | w1,w2) = Count(w1, w2, w3) / Count(w1, w2)
    # The denominator should be the count of the bigram (w1, w2)
    last_two_words_bigram_count = bigram_counts.get(last_two_words_tuple, 0)
    if last_two_words_bigram_count == 0:
        return f"'{' '.join(last_two_words_tuple)}' not found as a bigram. Cannot predict next word."

    predicted_word = None
    max_probability = -1

    for next_word, trigram_count in potential_next_words.items():
        probability = trigram_count / last_two_words_bigram_count
        print("probability of  " f'{next_word}' " is ", probability)
        if probability > max_probability:
            max_probability = probability
            predicted_word = next_word

    return predicted_word

# Example usage:
# Ensure bigram_counts, unigram_counts, and Trigrams_counts are defined from previous cells
# (They are present in the kernel state.)

# Try with a word sequence
sequence1 = "the movie was"
next_word1 = predict_next_word_trigram(sequence1, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

sequence2 = "the pizza was"
next_word2 = predict_next_word_trigram(sequence2, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")
```

```
probability of  was is  1.0
Given sequence: 'the movie', predicted next word: 'was'
probability of  was is  1.0
Given sequence: 'the pizza', predicted next word: 'was'
```

DEPLOYMENT OF TRI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_trigram(ip_text, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : a puppy from
probability of  the is  1.0
Given sequence: 'a puppy from', predicted next word: 'the'
```

NEXT WORD PREDICTION USING BI-GRAM MODEL WITH LAPLACE SMOOTHENING

```
def predict_next_word_bigram_Laplace(word_sequence, bigram_counts, unigram_counts):
    # Tokenize the input sequence and get the last word
    words_in_sequence = word_sequence.lower().split()
```

```
    if not words_in_sequence:
        return "Please provide a word sequence."
    last_word = words_in_sequence[-1]

    # Find potential next words based on bigrams starting with last_word
    potential_next_words = {}
    for (w1, w2), count in bigram_counts.items():
        if w1 == last_word:
            potential_next_words[w2] = count

    if not potential_next_words:
        return f"No bigram found starting with '{last_word}'."

    # Calculate probabilities for potential next words
    # P(w2 | w1) = Count(w1, w2) / Count(w1)
    last_word_unigram_count = unigram_counts.get(last_word, 0)
    if last_word_unigram_count == 0:
        return f"'{last_word}' not found in unigram counts. Cannot predict next word."

    predicted_word = None
    max_probability = -1

    for next_word, bigram_count in potential_next_words.items():
        probability = (bigram_count+1) / (last_word_unigram_count+V)
        print("probability of " f'{next_word}' " is ", probability)
        if probability > max_probability:
            max_probability = probability
            predicted_word = next_word

    return predicted_word

# Example usage:
# Ensure bigram_counts and unigram_counts are defined from previous cells
# (They are present in the kernel state.)

# Try with a word sequence
sequence1 = "a puppy"
next_word1 = predict_next_word_bigram_Laplace(sequence1, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

sequence2 = "the movie"
next_word2 = predict_next_word_bigram_Laplace(sequence2, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")

sequence3 = "my laptop"
next_word3 = predict_next_word_bigram_Laplace(sequence3, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence3}', predicted next word: '{next_word3}'")

sequence4 = "how are"
next_word4 = predict_next_word_bigram_Laplace(sequence4, bigram_counts, unigram_counts)
print(f"Given sequence: '{sequence4}', predicted next word: '{next_word4}'")
```

```
probability of from is  0.03773584905660377
Given sequence: 'a puppy', predicted next word: 'from'
probability of was is  0.03773584905660377
Given sequence: 'the movie', predicted next word: 'was'
probability of battery is  0.03773584905660377
Given sequence: 'my laptop', predicted next word: 'battery'
Given sequence: 'how are', predicted next word: 'No bigram found starting with 'are'.'
```

DEPLOYMENT OF LAPLACE SMOOTHENING BASED ON BI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_bigram_Laplace(ip_text, bigram_counts, unigram_counts)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : the rain
probability of caused is  0.03773584905660377
Given sequence: 'the rain', predicted next word: 'caused'
```

NEXT WORD PREDICTION USING TRI-GRAM MODEL BASED ON LAPLACE SMOOTHENING

```
def predict_next_word_trigram_Laplace(word_sequence, Trigrams_counts, bigram_counts):
    # Tokenize the input sequence
    words_in_sequence = word_sequence.lower().split()
    if not words_in_sequence:
```

```
        return "Please provide a word sequence."

    # Ensure at least two words for trigram prediction
    if len(words_in_sequence) < 2:
        return "Sequence must contain at least two words for trigram prediction."

    # Get the last two words as a tuple
    last_two_words_tuple = tuple(words_in_sequence[-2:])

    # Find potential next words based on trigrams starting with the last two words
    potential_next_words = {}
    for (w1, w2, w3), count in Trigrams_counts.items():
        if (w1, w2) == last_two_words_tuple:
            potential_next_words[w3] = count

    if not potential_next_words:
        return f"No trigram found starting with '{' '.join(last_two_words_tuple)}'."

    # Calculate probabilities for potential next words
    # P(w3 | w1,w2) = Count(w1, w2, w3) / Count(w1, w2)
    # The denominator should be the count of the bigram (w1, w2)
    last_two_words_bigram_count = bigram_counts.get(last_two_words_tuple, 0)
    if last_two_words_bigram_count == 0:
        return f"'{' '.join(last_two_words_tuple)}' not found as a bigram. Cannot predict next word."

    predicted_word = None
    max_probability = -1

    for next_word, trigram_count in potential_next_words.items():
        probability = (trigram_count+1) / (last_two_words_bigram_count+V)
        print("probability of  " f'{next_word}' " is ", probability)
        if probability > max_probability:
            max_probability = probability
            predicted_word = next_word

    return predicted_word

# Example usage:
# Ensure bigram_counts, unigram_counts, and Trigrams_counts are defined from previous cells
# (They are present in the kernel state.)

# Try with a word sequence
sequence1 = "the pizza was"
next_word1 = predict_next_word_trigram_Laplace(sequence1, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

sequence2 = "heavy rain is"
next_word2 = predict_next_word_trigram_Laplace(sequence2, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")
```

```
probability of  cold is  0.03773584905660377
Given sequence: 'the pizza was', predicted next word: 'cold'
Given sequence: 'heavy rain is', predicted next word: 'No trigram found starting with 'rain is'.'
```

DEPLOYMENT OF LAPLACE SMOOTHENING BASED ON TRI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_trigram_Laplace(ip_text, Trigrams_counts, bigram_counts)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : the exam was
probability of  more is  0.03773584905660377
Given sequence: 'the exam was', predicted next word: 'more'
```

NEXT WORD PREDICTION USING BI-GRAM MODEL ADD - K SMOOTHENING

```
def predict_next_word_bigram_K(word_sequence, bigram_counts, unigram_counts, K): #K=0.5-0.01
    # Tokenize the input sequence and get the last word
    words_in_sequence = word_sequence.lower().split()
    if not words_in_sequence:
        return "Please provide a word sequence."
    last_word = words_in_sequence[-1]

    # Find potential next words based on bigrams starting with last_word
    potential_next_words = {}
```

```
        for (w1, w2), count in bigram_counts.items():
            if w1 == last_word:
                potential_next_words[w2] = count

        if not potential_next_words:
            return f"No bigram found starting with '{last_word}'."

        # Calculate probabilities for potential next words
        # P(w2 | w1) = Count(w1, w2) / Count(w1)
        last_word_unigram_count = unigram_counts.get(last_word, 0)
        if last_word_unigram_count == 0:
            return f"'{last_word}' not found in unigram counts. Cannot predict next word."

        predicted_word = None
        max_probability = -1

        for next_word, bigram_count in potential_next_words.items():
            probability = (bigram_count+K) / (last_word_unigram_count+K*V)
            print("probability of " f'{next_word}' " is ", probability)
            if probability > max_probability:
                max_probability = probability
                predicted_word = next_word

        return predicted_word

    # Example usage:
    # Ensure bigram_counts and unigram_counts are defined from previous cells
    # (They are present in the kernel state.)

    # Try with a word sequence
    sequence1 = "heavy rain"
    next_word1 = predict_next_word_bigram_K(sequence1, bigram_counts, unigram_counts,0.5)
    print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

    sequence2 = "a puppy"
    next_word2 = predict_next_word_bigram_K(sequence2, bigram_counts, unigram_counts,0.5)
    print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")

    sequence3 = "the pizza"
    next_word3 = predict_next_word_bigram_K(sequence3, bigram_counts, unigram_counts,0.5)
    print(f"Given sequence: '{sequence3}', predicted next word: '{next_word3}'")

    sequence4 = "nlp is"
    next_word4 = predict_next_word_bigram_K(sequence4, bigram_counts, unigram_counts,0.5)
    print(f"Given sequence: '{sequence4}', predicted next word: '{next_word4}'")
```

```
probability of caused is  0.05555555555555555
Given sequence: 'heavy rain', predicted next word: 'caused'
probability of from is  0.05555555555555555
Given sequence: 'a puppy', predicted next word: 'from'
probability of was is  0.05555555555555555
Given sequence: 'the pizza', predicted next word: 'was'
Given sequence: 'nlp is', predicted next word: 'No bigram found starting with 'is'.'
```

DEPLOYMENT OF ADD-K SMOOTHENING BASED ON BI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_bigram_K(ip_text, bigram_counts, unigram_counts,0.5)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : heavy rain
probability of caused is  0.05555555555555555
Given sequence: 'heavy rain', predicted next word: 'caused'
```

NEXT WORD PREDICTION USING TRI-GRAM MODEL BASED ON ADD-K SMOOTHENING

```
def predict_next_word_trigram_K(word_sequence, Trigrams_counts, bigram_counts,K): #K=0.5-0.01
    # Tokenize the input sequence
    words_in_sequence = word_sequence.lower().split()
    if not words_in_sequence:
        return "Please provide a word sequence."

    # Ensure at least two words for trigram prediction
    if len(words_in_sequence) < 2:
        return "Sequence must contain at least two words for trigram prediction."
```

```
        # Get the last two words as a tuple
        last_two_words_tuple = tuple(words_in_sequence[-2:])

        # Find potential next words based on trigrams starting with the last two words
        potential_next_words = {}
        for (w1, w2, w3), count in Trigrams_counts.items():
            if (w1, w2) == last_two_words_tuple:
                potential_next_words[w3] = count

        if not potential_next_words:
            return f"No trigram found starting with '{' '.join(last_two_words_tuple)}'."

        # Calculate probabilities for potential next words
        # P(w3 | w1,w2) = Count(w1, w2, w3) / Count(w1, w2)
        # The denominator should be the count of the bigram (w1, w2)
        last_two_words_bigram_count = bigram_counts.get(last_two_words_tuple, 0)
        if last_two_words_bigram_count == 0:
            return f"'{' '.join(last_two_words_tuple)}' not found as a bigram. Cannot predict next word."

        predicted_word = None
        max_probability = -1

        for next_word, trigram_count in potential_next_words.items():
            probability = (trigram_count+K) / (last_two_words_bigram_count+K*V)
            print("probability of  " f'{next_word}' " is ", probability)
            if probability > max_probability:
                max_probability = probability
                predicted_word = next_word

        return predicted_word

    # Example usage:
    # Ensure bigram_counts, unigram_counts, and Trigrams_counts are defined from previous cells
    # (They are present in the kernel state.)

    # Try with a word sequence
    sequence1 = "the movie was"
    next_word1 = predict_next_word_trigram_K(sequence1, Trigrams_counts, bigram_counts,0.5)
    print(f"Given sequence: '{sequence1}', predicted next word: '{next_word1}'")

    sequence2 = "I have done"
    next_word2 = predict_next_word_trigram_K(sequence2, Trigrams_counts, bigram_counts,0.5)
    print(f"Given sequence: '{sequence2}', predicted next word: '{next_word2}'")
```

```
probability of  surprisingly is  0.05555555555555555
Given sequence: 'the movie was', predicted next word: 'surprisingly'
Given sequence: 'I have done', predicted next word: 'No trigram found starting with 'have done'.'
```

DEPLOYMENT OF ADD-K SMOOTHENING BASED ON TRI-GRAM MODEL

```
ip_text=input("Enter text : ")
next_word1 = predict_next_word_trigram_K(ip_text, Trigrams_counts, bigram_counts,0.5)
print(f"Given sequence: '{ip_text}', predicted next word: '{next_word1}'")
```

```
Enter text : the movie was
probability of  surprisingly is  0.05555555555555555
Given sequence: 'the movie was', predicted next word: 'surprisingly'
```