Batu Arda Düzgün
21802633

Part 1.a:

I downloaded the "Charles Dickens Processed.txt" from the Moodle.

Part 1.b:

I am using nltk version 3.7 and it takes around 10 minutes to tokenize the txt file. Because of this my code saves the tokens in a .txt files (one token each row) and reads from it on the subsequent rounds which takes 1 seconds.

Part 1.c:

The pos_tag function takes around 6 minutes to run because of this my code saves the tokens with their pos_tags in a .txt file and reads from it on the subsequent rounds which takes 2 seconds.

Part 1.d:

I updated the "custom lemmatizer.py" function to also lemmatize the verbs and adverbs. I use this function to iterate over the tokens with their pos_tags to lemmatize them.

Part 1.e:

I used 2 different for loops to create the bigrams with different window sizes. It is important note that the bigrams also have the pos_tag of the words at this point but they will be removed in the next step.

Part 1.f:

I deleted the bigrams from the list of bigrams form the previous part. The deleted bigrams which are deleted do not satisfy the one of the first 3 conditions for colocation candidates. Then I removed the pos_tags from the bigrams, and contend their frequencies and made a frequency dictionary for the bigrams and eliminated bigrams which accrued less then 10 times. I also created a frequency dictionary for the monograms which will be used in the next part.

Part 2.a:

I wrote functions to calculate the student's t-test, chi- square test, likelihood ratio test. These functions will be further discussed in part 3. One thing which is important to note is for the likelihood ratio test once each binomial probability is calculated I replace it with "math.ulp(0.0)" (around $5 \cdot 10^{-325}$) which is the smallest positive number in python. Then I take the log of each binomial probability and make the next computation with them only turning it back to normal at the end (10^{prob}).

I iterated over the colocation candidates for the two bigram dictionaries and compute the student's t-test, chi- square test, likelihood ratio test results and saved the result in a dictionary. Then I ordered the dictionaries and printed the first 20 elements of each test for each window size with the, bigram counts and individual word counts.

Batu Arda Düzgün
21802633

Part 3.a:

students t-test:

For the students t-test we use a null hypothesis of the two word in a bigram being independent from each other. The formula for students t-test is:

$$t = \frac{\bar{X} - \mu}{\sqrt{\frac{S^2}{N}}}$$

Where $\bar{X}$ is the mean of the test result, S^2 is the variance of the test result, μ is the mean of the null hypothesis and N is the total number of bigrams. $\bar{X}$ is $Pr(bigram) = \#w1w2\ /\ N$. S^2 is variance of a binomial number so it is equal to $Pr(bigram) \cdot (1 - Pr(bigram))$ but because $Pr(bigram)$ is very small we simply use $Pr(bigram)$. μ comes from the assumption of the bigram words being independent so it is simply equal to $\Pr(w1) \cdot \Pr(w2) = \#w1 \cdot \#w2/N^2$. Finally N = #tokens -1 and because there are 1898607 tokens it is estimated as #tokens.

Pearson's Chi-Square Test:

For the Pearson's Chi-Square test we use a null hypothesis of the two word in a bigram being independent from each other. The formula for Pearson's Chi-Square test is:

$$X^2 = \sum_{i,j} \frac{\left(O_{ij} - E_{ij}\right)^2}{E_{ij}}$$

where O_{ij} and E_{ij} are the observed vs expected counts of events i, j. However, there is a short cut for the 2-by-2 case. I used that and because of that I only needed to find O_{ij} values and not E_{ij} values. The formula for the short cut is:

$$X^2 = \sum_{i,j} \frac{\left(O_{ij} - E_{ij}\right)^2}{E_{ij}} = \frac{N(O_{11}O_{22} - O_{12}O_{21})^2}{(O_{11} + O_{12})(O_{11} + O_{21})(O_{12} + O_{22})(O_{21} + O_{22})}$$

The $O_{11} = \#w1w2$, $O_{12} = \#w2 - \#w1w2$, $O_{21} = \#w1 - \#w1w2$, and $O_{22} = N - \#w1 - \#w2 + \#w2w1$ are found by the given formulas and they can be represented with the table given bellow.

	w2	¬w2
w1	O_{11}	O_{12}
¬w1	O_{21}	O_{22}

Batu Arda Düzgün
21802633

Likelihood Ratios Test:

For likelihood ratio test, our Hypothesis 1 is the two words in the collation is independent so:

$$\Pr(\mathrm{w2} \mid \mathrm{w1}) = \Pr(\mathrm{w2} \mid \neg \mathrm{w1}) = \mathrm{p}$$

And our Hypothesis 2 the two words in the bigram is dependent so:

$$\Pr(\mathrm{w2} \mid \mathrm{w1}) = \mathrm{p1} \neq \Pr(\mathrm{w2} \mid \neg \mathrm{w1}) = \mathrm{p2}$$

Let $c_{12} = \#w1w2$, $c_1 = \#w1$, $c_2 = \#w2$

So for Hypothesis 1:

$$p = c_2/N$$

$$L(H1) = binom(c_{12}; c1, p) \cdot binom(c_2 - c_{12}; N - c_1, p)$$

So for Hypothesis 2:

$$p_1 = c_{12}/c_1$$

$$p_2 = \frac{c_2 - c_{12}}{N - c_1}$$

$$L(H2) = binom(c_{12}; c1, p_1) \cdot binom(c_2 - c_{12}; N - c_1, p_2)$$

Then the likelihood ratio score which is asymptotically $\chi 2$ distributed is:

$$-2 \cdot \ln\left(\frac{L(H1)}{L(H2)}\right)$$

Part 3.b:

In the Student's t-test threshold because there are too many dimensions of freedom we use the infinite dimension row.

In Pearson's Chi-Square Test threshold, we use a 2-by-2 matrix so our dimensions of freedom is (2-1) (2-1) = 1 and we use that row.

Likelihood Ratios Test threshold the scores are asymptotically $\chi 2$ distributed and because the difference of dimension of freedom between the two hypothesis is 1 we use the df=1 row of the table for Chi-Square.

Appendix:

```
import nltk
import time

import numpy as np

import custom_lemmatizer
from tabulate import tabulate

from scipy.stats import binom

import math

# nltk.download('wordnet')
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('universal_tagset')


print('The nltk version is {}.'.format(nltk.__version__))

f2 = open("Charles Dickens Processed.txt", 'r')

text = ""

for line in f2.readlines():
    text += line

f2.close()



# this is made to decrease run time
try:

    start_time = time.time()

    f1 = open("tokens.txt", 'r')

    tokens = []

    for token in f1.readlines():

        tokens.append(token[:-1])

    f1.close()

    print("--- %s seconds token reading time ---" % (time.time() -
start_time))

except:

    print("re creating the tokens list")

    start_time = time.time()
    tokens = nltk.word_tokenize(text)
```

Batu Arda Düzgün
21802633

```
    print("--- %s seconds tokenization time ---" % (time.time() - start_time))


    f3 = open("tokens.txt", 'w')

    for token in tokens:

        f3.write(token)
        f3.write("\n")

    f3.close()


try:
    start_time = time.time()
    f5 = open("pos_tag_tokens.txt", 'r')

    pos_tag_tokens = []

    for pos_tag_token in f5.readlines():
        temp = pos_tag_token[:-1]
        temp_pair = temp.split()
        pos_tag_tokens.append((temp_pair[0], temp_pair[1]))

    f5.close()
    print("--- %s seconds pos tag reading time ---" % (time.time() -
start_time))
except:

    print("re creating the tokens with pos tags list")

    start_time = time.time()
    pos_tag_tokens = nltk.pos_tag(tokens, tagset="universal")
    print("--- %s seconds pos tag time ---" % (time.time() - start_time))

    f4 = open("pos_tag_tokens.txt", 'w')

    for pos_tag_token in pos_tag_tokens:
        f4.write(pos_tag_token[0] + " " + pos_tag_token[1])
        f4.write("\n")

    f4.close()


cm = custom_lemmatizer.custom_lemmatizer()

lemmaized_tokens = []

start_time = time.time()
for t in pos_tag_tokens:

    lemmaized_tokens.append((cm.lemmatize(t), t[1]))

print("--- %s seconds lemmatization time ---" % (time.time() - start_time))


start_time = time.time()
```

```
bigrams_1 = []

for i in range(len(lemmaized_tokens) - 1):

    bigrams_1.append((lemmaized_tokens[i], lemmaized_tokens[i + 1]))


bigrams_2 = []

for i in range(len(lemmaized_tokens) - 3):

    bigrams_2.append((lemmaized_tokens[i], lemmaized_tokens[i + 1]))
    bigrams_2.append((lemmaized_tokens[i], lemmaized_tokens[i + 2]))
    bigrams_2.append((lemmaized_tokens[i], lemmaized_tokens[i + 3]))

bigrams_2.append((lemmaized_tokens[-3], lemmaized_tokens[-2]))
bigrams_2.append((lemmaized_tokens[-3], lemmaized_tokens[-1]))
bigrams_2.append((lemmaized_tokens[-2], lemmaized_tokens[-1]))

print("--- %s seconds bigram creation time ---" % (time.time() - start_time))


def CountFrequency(my_list):
    # Creating an empty dictionary
    freq = {}
    for item in my_list:
        if (item in freq):
            freq[item] += 1
        else:
            freq[item] = 1

    return freq

start_time = time.time()

#clearing for pos tags
bigrams_to_keep_1 = []

for i in range(len(bigrams_1)):

    if(((bigrams_1[i][0][1] == "NOUN") or (bigrams_1[i][0][1] == "ADJ")) and
(bigrams_1[i][1][1] == "NOUN")):

        bigrams_to_keep_1.append(bigrams_1[i])

bigrams_1 = bigrams_to_keep_1

#clearing for punctuation
bigrams_to_keep_1 = []

for i in range(len(bigrams_1)):

    if((bigrams_1[i][0][0].isalpha()) and (bigrams_1[i][1][0].isalpha())):

        bigrams_to_keep_1.append((bigrams_1[i][0][0], bigrams_1[i][1][0]))
```

Batu Arda Düzgün
21802633

```
bigrams_1 = bigrams_to_keep_1



stopwords = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves",
"you", "your", "yours", "yourself", "yourselves", "he", "him", "his",
"himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",
"them", "their", "theirs", "themselves", "what", "which", "who", "whom",
"this", "that", "these", "those", "am", "is", "are", "was", "were", "be",
"been", "being", "have", "has", "had", "having", "do", "does", "did", "doing",
"a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while",
"of", "at", "by", "for", "with", "about", "against", "between", "into",
"through", "during", "before", "after", "above", "below", "to", "from", "up",
"down", "in", "out", "on", "off", "over", "under", "again", "further", "then",
"once", "here", "there", "when", "where", "why", "how", "all", "any", "both",
"each", "few", "more", "most", "other", "some", "such", "no", "nor", "not",
"only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will",
"just", "don", "should", "now"]

#clearing the stopwords
bigrams_to_keep_1 = []

for i in range(len(bigrams_1)):

    if((bigrams_1[i][0] not in stopwords) and (bigrams_1[i][1] not in
stopwords)):

        bigrams_to_keep_1.append(bigrams_1[i])

bigrams_1 = bigrams_to_keep_1

# deleting less then 10 bigrams
temp_dictinory = CountFrequency(bigrams_1)
bigram_dictionary_1 = {k: v for k, v in sorted(temp_dictinory.items(),
key=lambda item: item[1], reverse=True)}

bigram_dictionary_to_keep = {}

part_e_bigram_dictionary_1 = bigram_dictionary_1

for key in bigram_dictionary_1:

    if bigram_dictionary_1[key] >= 10:

        bigram_dictionary_to_keep[key] = bigram_dictionary_1[key]

    else:

        break

bigram_dictionary_1 = bigram_dictionary_to_keep




#clearing for pos tags
bigrams_to_keep_2 = []
```

Batu Arda Düzgün
21802633

```
for i in range(len(bigrams_2)):

    if(((bigrams_2[i][0][1] == "NOUN") or (bigrams_2[i][0][1] == "ADJ")) and
(bigrams_2[i][1][1] == "NOUN")):

        bigrams_to_keep_2.append(bigrams_2[i])

bigrams_2 = bigrams_to_keep_2

#clearing for punctuation
bigrams_to_keep_2 = []

for i in range(len(bigrams_2)):

    if((bigrams_2[i][0][0].isalpha()) and (bigrams_2[i][1][0].isalpha())):

        bigrams_to_keep_2.append((bigrams_2[i][0][0], bigrams_2[i][1][0]))

bigrams_2 = bigrams_to_keep_2



stopwords = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves",
"you", "your", "yours", "yourself", "yourselves", "he", "him", "his",
"himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",
"them", "their", "theirs", "themselves", "what", "which", "who", "whom",
"this", "that", "these", "those", "am", "is", "are", "was", "were", "be",
"been", "being", "have", "has", "had", "having", "do", "does", "did", "doing",
"a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while",
"of", "at", "by", "for", "with", "about", "against", "between", "into",
"through", "during", "before", "after", "above", "below", "to", "from", "up",
"down", "in", "out", "on", "off", "over", "under", "again", "further", "then",
"once", "here", "there", "when", "where", "why", "how", "all", "any", "both",
"each", "few", "more", "most", "other", "some", "such", "no", "nor", "not",
"only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will",
"just", "don", "should", "now"]

#clearing the stopwords
bigrams_to_keep_2 = []

for i in range(len(bigrams_2)):

    if((bigrams_2[i][0] not in stopwords) and (bigrams_2[i][1] not in
stopwords)):

        bigrams_to_keep_2.append(bigrams_2[i])

bigrams_2 = bigrams_to_keep_2

# deleting less then 10 bigrams
temp_dictinory = CountFrequency(bigrams_2)
bigram_dictionary_2 = {k: v for k, v in sorted(temp_dictinory.items(),
key=lambda item: item[1], reverse=True)}

bigram_dictionary_to_keep = {}
```

```
part_e_bigram_dictionary_2 = bigram_dictionary_2

for key in bigram_dictionary_2:

    if bigram_dictionary_2[key] >= 10:

        bigram_dictionary_to_keep[key] = bigram_dictionary_2[key]

    else:

        break

bigram_dictionary_2 = bigram_dictionary_to_keep

print("--- %s seconds bigram culling time ---" % (time.time() - start_time))


clean_lemmaized_tokens = ["a"] * len(lemmaized_tokens)

for i in range(len(lemmaized_tokens)):

    clean_lemmaized_tokens[i] = lemmaized_tokens[i][0]


monogram_dictionary = CountFrequency(clean_lemmaized_tokens)

token_count = len(lemmaized_tokens)


# part 1.b

print("\npart 1.b:")
print(f"N = {token_count}")

print("\npart 1.d:")
print(f"that = {monogram_dictionary['that']}")
print(f"the = {monogram_dictionary['the']}")
print(f"negligent = {monogram_dictionary['negligent']}")
print(f"london = {monogram_dictionary['london']}")
print(f". = {monogram_dictionary['.']}")

print("\npart 1.e:")
print(f"(“fashionable”,“intelligence”) in windows of size 1 =
{part_e_bigram_dictionary_1[('fashionable','intelligence')]}")
print(f"(“high”,“chancery”) in windows of size 3 =
{part_e_bigram_dictionary_2[('high','chancery')]}")

print("\npart 1.f:")
if ("mr.","skimpole") in bigram_dictionary_1.keys():
    answer = "yes"
else:
    answer = "No"
print(f"(“mr.”,“skimpole”) is in windows of size 1 = {answer}")
if ("spontaneous","combustion") in bigram_dictionary_2.keys():
    answer = "yes"
else:
    answer = "No"
```

```
print(f"("spontaneous","combustion") is in windows of size 3 = {answer}")


print("\n\npart 2.a:\n")

# for window size 1

# students' t test:

temp_dictinory = {}

for key in bigram_dictionary_1:

    H0 = monogram_dictionary[key[0]] * monogram_dictionary[key[1]] /
(token_count * token_count)
    MLE = bigram_dictionary_1[key] / token_count

    t = (MLE - H0) / np.sqrt(MLE / token_count)

    temp_dictinory[key] = t


t_test_dictionary1 = {k: v for k, v in sorted(temp_dictinory.items(),
key=lambda item: item[1], reverse=True)}

# t test table

table_content = []
rank = 1

for key in t_test_dictionary1:

    name = key[0] + " " + key[1]

    result_row = [rank, name, t_test_dictionary1[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]

    table_content.append(result_row)

    rank += 1

    if rank > 20:
        break



print("\n\nStudent's t test for window size 1")
print(tabulate(table_content, headers=['Rank', 'Bigram', "t-score", "c(w1w2)",
"c(w1)", "c(w2)"]))
print("\n")



# Pearson's Chi-Square test:
```

```
temp_dictinory = {}

for key in bigram_dictionary_1:

    O_11 = bigram_dictionary_1[key]
    O_12 = monogram_dictionary[key[0]] - O_11
    O_21 = monogram_dictionary[key[1]] - O_11
    O_22 = token_count - O_21 - O_12 - O_11

    X2 = token_count * (O_11 * O_22 - O_12 * O_21) ** 2 / ((O_11 + O_12) *
(O_11 + O_21) * (O_12 + O_22) * (O_21 + O_22))

    temp_dictinory[key] = X2


Pearson_Chi_Square_test_dictionary1 = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}

# Pearson’s Chi-Square test table

table_content = []
rank = 1

for key in Pearson_Chi_Square_test_dictionary1:

    name = key[0] + " " + key[1]

    result_row = [rank, name, Pearson_Chi_Square_test_dictionary1[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]

    table_content.append(result_row)

    rank += 1

    if rank > 20:
        break


print("\n\nPearson’s Chi-Square test for window size 1")
print(tabulate(table_content, headers=['Rank', 'Bigram', "X^2 score",
"c(w1w2)", "c(w1)", "c(w2)"]))
print("\n")






# Likelihood ratio:

temp_dictinory = {}

for key in bigram_dictionary_1:

    c_12 = bigram_dictionary_1[key]
    c_1 = monogram_dictionary[key[0]]
```

Batu Arda Düzgün
21802633

```
    c_2 = monogram_dictionary[key[1]]
    N = token_count

    H1_p = c_2 / N

    H2_p1 = c_12 / c_1
    H2_p2 = (c_2 - c_12) / (N - c_1)



    H1_term_1 = binom.pmf(c_12, c_1, H1_p)
    if H1_term_1 == 0:
        H1_term_1 = math.ulp(0.0)

    H1_term_2 = binom.pmf(c_2-c_12, N-c_1, H1_p)
    if H1_term_2 == 0:
        H1_term_2 = math.ulp(0.0)

    H2_term_1 = binom.pmf(c_12, c_1, H2_p1)
    if H2_term_1 == 0:
        H2_term_1 = math.ulp(0.0)

    H2_term_2 = binom.pmf(c_2-c_12, N-c_1, H2_p2)
    if H2_term_2 == 0:
        H2_term_2 = math.ulp(0.0)

    log_L_H1 = math.log(H1_term_1) + math.log(H1_term_2)

    log_L_H2 = math.log(H2_term_1) + math.log(H2_term_2)


    log_likelihood_ratio = log_L_H1 - log_L_H2
    likelihood_ratio_score = -2 * log_likelihood_ratio

    temp_dictinory[key] = likelihood_ratio_score


likelihood_ratio_test_dictionary1 = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}


# likelihood ratio test table

table_content = []
rank = 1

for key in likelihood_ratio_test_dictionary1:

    name = key[0] + " " + key[1]

    result_row = [rank, name, likelihood_ratio_test_dictionary1[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]

    table_content.append(result_row)

    rank += 1
```

```
    if rank > 20:
        break


print("\n\n Likelihood Ratio Test for window size 1")
print(tabulate(table_content, headers=['Rank', 'Bigram', "Likelihood Ratio",
"c(w1w2)", "c(w1)", "c(w2)"]))
print("\n")







# for window size 3

# adjusting for window size 3

token_count *= 3

for key in monogram_dictionary:
    monogram_dictionary[key] *= 3



# students' t test:

temp_dictinory = {}

for key in bigram_dictionary_2:

    H0 = monogram_dictionary[key[0]] * monogram_dictionary[key[1]] /
(token_count * token_count)
    MLE = bigram_dictionary_2[key] / token_count

    t = (MLE - H0) / np.sqrt(MLE / token_count)

    temp_dictinory[key] = t


t_test_dictionary = {k: v for k, v in sorted(temp_dictinory.items(),
key=lambda item: item[1], reverse=True)}

# t test table

table_content = []
rank = 1

for key in t_test_dictionary:

    name = key[0] + " " + key[1]

    result_row = [rank, name, t_test_dictionary[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]
```

```
    table_content.append(result_row)

    rank += 1

    if rank > 20:
        break



print("\n\nStudent's t test for window size 3")
print(tabulate(table_content, headers=['Rank', 'Bigram', "t-score", "c(w1w2)",
"c(w1)", "c(w2)"]))
print("\n")



# Pearson’s Chi-Square test:

temp_dictinory = {}

for key in bigram_dictionary_2:

    O_11 = bigram_dictionary_2[key]
    O_12 = monogram_dictionary[key[0]] - O_11
    O_21 = monogram_dictionary[key[1]] - O_11
    O_22 = token_count - O_21 - O_12 - O_11

    X2 = token_count * (O_11 * O_22 - O_12 * O_21) ** 2 / ((O_11 + O_12) *
(O_11 + O_21) * (O_12 + O_22) * (O_21 + O_22))

    temp_dictinory[key] = X2


Pearson_Chi_Square_test_dictionary = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}

# Pearson’s Chi-Square test table

table_content = []
rank = 1

for key in Pearson_Chi_Square_test_dictionary:

    name = key[0] + " " + key[1]

    result_row = [rank, name, Pearson_Chi_Square_test_dictionary[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]

    table_content.append(result_row)

    rank += 1

    if rank > 20:
        break
```

```
print("\n\nPearson’s Chi-Square test for window size 3")
print(tabulate(table_content, headers=['Rank', 'Bigram', "X^2 score",
"c(w1w2)", "c(w1)", "c(w2)"]))
print("\n")





# Likelihood ratio:

temp_dictinory = {}

for key in bigram_dictionary_2:

    c_12 = bigram_dictionary_2[key]
    c_1 = monogram_dictionary[key[0]]
    c_2 = monogram_dictionary[key[1]]
    N = token_count

    H1_p = c_2 / N

    H2_p1 = c_12 / c_1
    H2_p2 = (c_2 - c_12) / (N - c_1)



    H1_term_1 = binom.pmf(c_12, c_1, H1_p)
    if H1_term_1 == 0:
        H1_term_1 = math.ulp(0.0)

    H1_term_2 = binom.pmf(c_2-c_12, N-c_1, H1_p)
    if H1_term_2 == 0:
        H1_term_2 = math.ulp(0.0)

    H2_term_1 = binom.pmf(c_12, c_1, H2_p1)
    if H2_term_1 == 0:
        H2_term_1 = math.ulp(0.0)

    H2_term_2 = binom.pmf(c_2-c_12, N-c_1, H2_p2)
    if H2_term_2 == 0:
        H2_term_2 = math.ulp(0.0)

    log_L_H1 = math.log(H1_term_1) + math.log(H1_term_2)

    log_L_H2 = math.log(H2_term_1) + math.log(H2_term_2)

    log_likelihood_ratio = log_L_H1 - log_L_H2
    likelihood_ratio_score = -2 * log_likelihood_ratio

    temp_dictinory[key] = likelihood_ratio_score

likelihood_ratio_test_dictionary = {k: v for k, v in
sorted(temp_dictinory.items(), key=lambda item: item[1], reverse=True)}
```

```
# likelihood ratio test table

table_content = []
rank = 1

for key in likelihood_ratio_test_dictionary:

    name = key[0] + " " + key[1]

    result_row = [rank, name, likelihood_ratio_test_dictionary[key],
bigram_dictionary_2[key], monogram_dictionary[key[0]],
monogram_dictionary[key[1]]]

    table_content.append(result_row)

    rank += 1

    if rank > 20:
        break


print("\n\n Likelihood Ratio Test for window size 3")
print(tabulate(table_content, headers=['Rank', 'Bigram', "Likelihood Ratio",
"c(w1w2)", "c(w1)", "c(w2)"]))
print("\n")


print("\n\npart 3.a:\n")
bigram = ("cursitor","street")
table_content=[["t-Test", t_test_dictionary1[bigram]],
                ["Chi-square Test",
Pearson_Chi_Square_test_dictionary1[bigram]],
                ["Likelihood Ratio Test",
likelihood_ratio_test_dictionary1[bigram]]]

print("Result table for ('cursitor','street') with window size 1")
print(tabulate(table_content, headers=['Test', 'Score']))
print("\n")

bigram = ("good","one")
table_content=[["t-Test", t_test_dictionary1[bigram]],
                ["Chi-square Test",
Pearson_Chi_Square_test_dictionary1[bigram]],
                ["Likelihood Ratio Test",
likelihood_ratio_test_dictionary1[bigram]]]

print("Result table for ('good','one') with window size 1")
print(tabulate(table_content, headers=['Test', 'Score']))
print("\n")
```

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

Batu Arda Düzgün
21802633

```
class custom_lemmatizer:

    tag_dict = {"ADJ": wordnet.ADJ,
                "NOUN": wordnet.NOUN,
                "VERB": wordnet.VERB,
                "ADV": wordnet.ADV}

    lemmatizer = WordNetLemmatizer()

    def lemmatize(self, word_pos_tuple):
        word = word_pos_tuple[0]
        pos_tag = word_pos_tuple[1]
        if pos_tag in self.tag_dict:
            return self.lemmatizer.lemmatize(word,
self.tag_dict[pos_tag]).lower()
        else:
            return word.lower()
```