EEE 486 Assignment 1 Report
The Hunt for Collocations

Emine İrem Esendemir
ID: 22202103

Bilkent University

Instructor: Aykut Koç

March 16, 2025

1 Introduction

Collocations are word pairs that frequently appear together, forming meaningful units in language. This assignment analyzes collocations in Jane Austen's novels using three statistical methods: Student's t test, Chi-square test, and likelihood ratio test.

The process includes tokenization, POS tagging, and lemmatization. Bigrams are filtered based on some constraints, such as part-of-speech tag patterns, stopword removal, and frequency thresholds. Statistical tests rank collocations based on their scores. Also, specific bigrams are tested for statistical significance.

2 Corpus Preprocessing

In this section, the provided corpus was preprocessed before extracting collocations so that the test scores can be calculated and reported only on these filtered bigrams. First, the text was read from the file and tokenized using nltk version 3.9.1. Then, part-of-speech tagging was assigned to these tokens with the universal tagset to help filter valid collocations. Subsequently, lemmatization was performed using `WordNetLemmatizer` to standardize word forms. In the lemmatization process, only the adjective and noun tokens are lemmatized because the candidate collocations are extracted from only these types of words.

The bigrams were extracted using two window sizes: a size of 1, where the second word is the immediate right neighbor, and a size of 3, where the second word is chosen from the next three words. The bigrams are kept in the format *((w1, tag1), (w2, tag2))*.

To filter collocations, only `NOUN-NOUN` and `ADJ-NOUN` bigrams were kept. The stopwords, according to the given list, were removed. Bigrams containing punctuation were discarded, and those occurring fewer than 10 times were filtered out. The final preprocessed data set consists of all bigrams with their counts, word counts to use in the tests, and candidate collocations, all for each window size. [1].

3 Finding the Collocations

After preprocessing, collocation candidates are evaluated using three statistical hypothesis tests: **Student's t-test, Chi-square test, and the Likelihood Ratio Test**. Each method calculates a score for bigrams, and ranks them based on their significance. The primary goal of these calculations is to determine whether the co-occurrence of words happens randomly. If it is not random, the words should appear together more frequently than expected based on chance.

1. Student's t-test

In the Student's t-test, it is assumed that the probability distribution approaches a Gaussian distribution as N (total number of tokens) goes to infinity. First, the null hypothesis H_0 (taken as the population mean μ)is formulated by assuming that the occurrences of the words are independent. Then, the Maximum Likelihood Estimate (MLE) (taken as the sample mean $\bar{X}$) is calculated based on the count of the collocation and the size of the corpus.

$$H_0 = \frac{c(w_1)}{N} \cdot \frac{c(w_2)}{N}$$

$$\text{MLE} = \frac{c(w_1 w_2)}{N}$$

In the t-score calculations, the MLE is considered the sample mean ($\bar{X}$), while H_0 is taken as the mean value (μ). In the Bernoulli distribution, the sample mean (MLE) is equal to p, and the sample varience S^2 is given by:

$$S^2 = p(1-p)$$

Since p is very small, this equation is approximately equal to p. The t-score is then computed as:

$$t = \frac{\bar{X} - \mu}{\sqrt{\frac{S^2}{N}}} = \frac{\text{MLE} - H_0}{\sqrt{\frac{\text{MLE}}{N}}}$$

A higher t score suggests stronger collocation. Using this formula, the t scores are calculated and the collocations are ranked accordingly.

2. Chi-square Test

The chi-square test evaluates statistical dependence by comparing observed and expected frequencies. In this test, O_{ij} represents the observed frequencies in the contingency table, while E_{ij} denotes the expected frequencies assuming independence. The total number of word occurrences is given by N.

The observed contingency table is:

$$\begin{bmatrix} c(w_1 w_2) & c(w_2) - c(w_1 w_2) \\ c(w_1) - c(w_1 w_2) & N - (c(w_1) + c(w_2) - c(w_1 w_2)) \end{bmatrix}$$

where $c(w_1 w_2)$ is the bigram count, and $c(w_1)$ and $c(w_2)$ are individual word counts. The chi-square score is then computed as:

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

As in the t-test, after calculating the chi-square scores, the collocations are sorted according to the scores, and the results are reported in the answer sheet. A higher chi-square score indicates a stronger word association.

3. Likelihood Ratio Test (LLR)

This test is done by comparing two hypotheses: the null hypothesis H_0, which assumes independence ($P(w_2|w_1) = P(w_2|\neg w_1)$), and the alternative hypothesis H_1, which assumes dependence ($P(w_2|w_1) \neq P(w_2|\neg w_1)$).

The likelihood of the data for each hypothesis is estimated using binomial distribution. The threshold for the binomial results is set as 10^{-100} to prevent $\log(0)$. When probabilities are lower than this threshold, their values are adjusted to the this value. Maximum likelihood estimates (MLEs) for the probabilities are obtained from corpus counts. Given a corpus with N bigrams, where c_1 and c_2 are the counts of words w_1 and w_2, and c_{12} is the count of their co-occurrence, the likelihoods under both hypotheses are computed as:

$$L(H_0) = b(c_{12}; c_1, p) \cdot b(c_2 - c_{12}; N - c_1, p)$$

$$L(H_1) = b(c_{12}; c_1, p_1) \cdot b(c_2 - c_{12}; N - c_1, p_2)$$

The log-likelihood ratio is then calculated as:

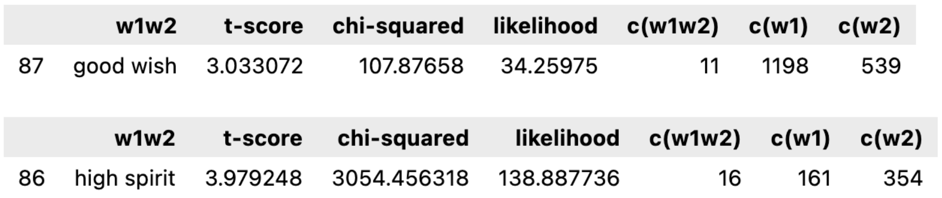
$$\text{LR} = -2 \times (\log L(H_0) - \log L(H_1))$$

Since $-2\log\lambda$ follows an asymptotic chi-square distribution, it can be tested for statistical significance. A higher log-likelihood ratio suggests stronger collocation strength.

This method is particularly useful for rare bigrams where other statistical tests, such as the chi-square test, may not perform well due to sparsity. Like the previous methods, collocations are ranked based on their LLR scores, and results are reported accordingly.

4 Explaining the Statistical Tests

It is asked to decide whether the bigrams *good wish* and *high spirit* are collocations with a significance value of $\alpha = 0.005$. The statistical tests are applied to these bigrams using a collocation window size of 1. Below, the test calculations and threshold values are explained in detail.



	w1w2	t-score	chi-squared	likelihood	c(w1w2)	c(w1)	c(w2)
87	good wish	3.033072	107.87658	34.25975	11	1198	539

	w1w2	t-score	chi-squared	likelihood	c(w1w2)	c(w1)	c(w2)
86	high spirit	3.979248	3054.456318	138.887736	16	161	354

Figure 1: Result of a Statistical Tests

1. Student's t-test Calculation

For **good wish**:

$$H_0 = \frac{1198}{686618} \times \frac{539}{686618} = \frac{645922}{4.717 \times 10^{11}} = 1.37 \times 10^{-6}$$

$$\text{MLE} = \frac{11}{686618} = 1.6 \times 10^{-5}$$

$$t = \frac{(1.6 \times 10^{-5}) - (1.37 \times 10^{-6})}{\sqrt{\frac{1.6 \times 10^{-5}}{686618}}} = 3.033072$$

For **high spirit**:

$$H_0 = \frac{161}{686618} \times \frac{354}{686618} = \frac{56994}{4.717 \times 10^{11}} = 1.21 \times 10^{-7}$$

$$\text{MLE} = \frac{16}{686618} = 2.33 \times 10^{-5}$$

$$t = \frac{(2.33 \times 10^{-5}) - (1.21 \times 10^{-7})}{\sqrt{\frac{2.33 \times 10^{-5}}{686618}}} = 3.979248$$

From the t-distribution table given in the assignment, the threshold for $\alpha = 0.005$ is **2.576**. Since both t-scores exceed this threshold, both bigrams qualify as collocations under the t-test.

2. Chi-square Test Calculation

The chi-square test statistic is calculated as:

$$\chi^2 = \frac{N(O_{11}O_{22} - O_{12}O_{21})^2}{(O_{11} + O_{12})(O_{11} + O_{21})(O_{12} + O_{22})(O_{21} + O_{22})}$$

where:

For **good wish**:

$$\begin{bmatrix} 11 & 539 - 11 = 528 \\ 1198 - 11 = 1187 & 686618 - (1198 + 539 - 11) = 684892) \end{bmatrix}$$

$$\chi^2 = \frac{686618(11 \times 684892 - 528 \times 1187)^2}{(11+528)(11+1187)(528+684892)(1187+684892)}$$

$$\chi^2 = 107.87658$$

For **high spirit**:

$$\begin{bmatrix} 16 & 354 - 16 = 338 \\ 161 - 16 = 145 & 686618 - (161 + 354 - 16) = 686119 \end{bmatrix}$$

$$\chi^2 = \frac{686618(16 \times 686119 - 338 \times 145)^2}{(16+338)(16+145)(338+686119)(145+686119)}$$

$$\chi^2 = 3054.456318$$

From the given chi-square table, the threshold at $\alpha = 0.005$ is **7.879** at n=1, n is 1 as *(row - 1) * (column - 1)* = 1. Since both values are significantly higher, both bigrams are counted as collocations under the chi-square test.

3. Likelihood Ratio Test Calculation

For **good wish**:

$$p = \frac{539}{686618} = 7.85 \times 10^{-4}$$

$$p_1 = \frac{11}{1198} = 9.18 \times 10^{-3}, \quad p_2 = \frac{528}{685420} = 7.7 \times 10^{-4}$$

$$\text{LR} = -2 \times \big[\log b(11; 1198, 7.85 \times 10^{-4}) + \log b(528; 685420, 7.85 \times 10^{-4})$$

$$- \log b(11; 1198, 9.18 \times 10^{-3}) - \log b(528; 685420, 7.7 \times 10^{-4})\big]$$

$$\text{LR} = 34.25975$$

For **high spirit**:

$$p = \frac{354}{686618} = 5.16 \times 10^{-4}$$

$$p_1 = \frac{16}{161} = 9.94 \times 10^{-2}, \quad p_2 = \frac{338}{686457} = 4.92 \times 10^{-4}$$

$$\text{LR} = -2 \times \Big[\log b(16; 161, 5.16 \times 10^{-4}) + \log b(338; 686457, 5.16 \times 10^{-4})$$

$$- \log b(16; 161, 9.94 \times 10^{-2}) - \log b(338; 686457, 4.92 \times 10^{-4})\Big]$$

$$\text{LR} = 138.887736$$

From the chi-square table, the threshold at $\alpha = 0.005$ is **7.879**. Since both likelihood ratio scores exceed this threshold, both bigrams are counted as collocations under the Likelihood Ratio Test.

5 Conclusion

In this study, statistical tests were applied to bigrams extracted from a corpus to identify collocations. The results showed that the **t-test** chooses frequently occurring collocations, while the **chi-square test** identifies rare but statistically significant word pairs. The **likelihood ratio test** detects collocations where individual word frequencies are high but the bigram itself is not that much high itself.

Overall, all three methods provided valuable insights, with some variation in detected collocations. The results confirmed that statistical measures can effectively differentiate meaningful word associations from random co-occurrences.

References

[1] NLTK GitHub Repository.

Appendix

```
### Part 1: Data Preporcessing
#imports
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
```

```
# necessary data is downloaded
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('universal_tagset')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
# reading the file
corpora_file = open("Jane Austen Processed.txt", "r")
corpora = corpora_file.read()
corpora_file.close()
```

```
# tokenize the text
tokens = word_tokenize(corpora)
```

```
# part 1.b
N = len(tokens)
print("Number of tokens in the corpus (N):", N);
```

```
# tokenized text
pos_tags = pos_tag(tokens, tagset='universal')
```

```
# lemmatizing the tokens based on the pos tags
def lemmatize_tokens(pos_tags):
    tag_dict = {"ADJ": wordnet.ADJ,
                "NOUN": wordnet.NOUN}
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = []
    for word, pos_tag in pos_tags:
        if pos_tag in tag_dict:
            lemmatized_token = lemmatizer.lemmatize(word,
              tag_dict[pos_tag]).lower()
        else:
            lemmatized_token = word.lower()
        lemmatized_tokens.append((lemmatized_token, pos_tag)
          )
    return lemmatized_tokens
```

```
# lemmatized tokens
lemmatized_tokens = lemmatize_tokens(pos_tags)
```

```
# part 1.d,
part1d_tokens = ["that","the","london","honor","."]
#the counts of the query_tokens words in the
   lemmatized_tokens
word_counts = {word: 0 for word in part1d_tokens}
for word, _ in lemmatized_tokens:
    if word in word_counts:
        word_counts[word] += 1

print("The counts of the part 1.d words are:\n")
for word, count in word_counts.items():
    print(f"{word}: {count}")
```

```
# all bigrams are found based on the given window size, with
    their POS tags
def find_all_bigrams(lemmatized_tokens, window_size):
    bigrams = []
    for i in range(len(lemmatized_tokens)):
        word_1_pair = lemmatized_tokens[i]
        window = lemmatized_tokens[i+1:i+1+window_size]
        for word_pair in window:
            bigram = (word_1_pair, word_pair)
            bigrams.append(bigram)
    return bigrams
```

```
# filtering the bigrams by their pos tag
def filter_by_tag(bigrams):
    candidate_collocations = []
    for bigram in bigrams:
        if (bigram[0][1] == "NOUN" or bigram[0][1] == "ADJ")
            and bigram[1][1] == "NOUN":
            candidate_collocation = (bigram[0][0], bigram
                [1][0])
            candidate_collocations.append(
                candidate_collocation)
    return candidate_collocations
```

```
# eliminate bigrams including stopwords (https://gist.github
    .com/sebleier/554280).
def get_stopwords():
    stop_words_file = open("nltk_list_of_english_stopwords.
        txt","r", encoding='utf-8-sig')
    stop_words_list = [stop_word.strip() for stop_word in
        stop_words_file]
    stop_words_file.close()
    return stop_words_list
```

```
# filtering the bigrams by the stop words
def filter_by_stopwords(bigrams, stopwords):
    candidate_collocations = []
    for bigram in bigrams:
        if bigram[0] not in stopwords and bigram[1] not in
            stopwords:
            candidate_collocation = bigram
            candidate_collocations.append(
                candidate_collocation)
    return candidate_collocations
```

```
# filtering the bigrams by the punctuation_marks
def filter_by_punctuation_marks(bigrams):
    candidate_collocations = []
    for word1, word2 in bigrams:
        if word1.isalpha() and word2.isalpha():
            bigram = (word1, word2)
            candidate_collocations.append(bigram)
    return candidate_collocations
```

```
# filtering the bigrams by their occurance number
def filter_by_occurance(bigrams, min_occurance):
    bigram_counts = {}
    # Count occurrences
    for bigram in bigrams:
        if bigram not in bigram_counts:
            bigram_counts[bigram] = 1
        else:
            bigram_counts[bigram] += 1

    # Filter bigrams by min_occurance
    candidate_collocations = {}
    for bigram, occurance in bigram_counts.items():
        if occurance >= min_occurance:
            candidate_collocations[bigram] = occurance
    return candidate_collocations.keys(),
       candidate_collocations
```

```
# get the word count of the bigrams
def get_word_counts(bigrams):
    word_counts = {"word1":{}, "word2":{}}
    for ((word1, pos_tag1), (word2, pos_tag2)) in bigrams:
        if word1 not in word_counts["word1"]:
            word_counts["word1"][word1] = 1
        else:
            word_counts["word1"][word1] += 1

        if word2 not in word_counts["word2"]:
            word_counts["word2"][word2] = 1
        else:
            word_counts["word2"][word2] += 1
    return word_counts
```

```
# candidate collocations, bigram counts and word counts are
   found for the given window sizes
window_sizes = [1,3]
collocations_dict = {}
stopwords = get_stopwords()
for window_size in window_sizes:
    bigrams = find_all_bigrams(lemmatized_tokens,
       window_size) #print(f"Number of bigrams for window
```

```
        size {window_size}:", len(bigrams))
    candidate_collocations = filter_by_tag(bigrams)
    candidate_collocations = filter_by_stopwords(
        candidate_collocations, stopwords)
    candidate_collocations = filter_by_punctuation_marks(
        candidate_collocations)
    candidate_collocations, bigram_counts =
        filter_by_occurance(candidate_collocations,
        min_occurance=10)
    word_counts = get_word_counts(bigrams)
    collocations_dict[window_size] = {
        "candidate_collocations": candidate_collocations,
        "bigram_counts": bigram_counts,
        "word_counts": word_counts,
        "bigrams": bigrams
    }
```

```
#part 1.e
window_1_bigrams = collocations_dict[1]["bigrams"]
window_1_bigrams_without_pos_tags = []
for ((word_1, pos_tag_1),(word_2, pos_tag_2)) in
    window_1_bigrams:
    window_1_bigrams_without_pos_tags.append((word_1, word_2
        ))

asked_bigram = ("good", "company")
print("Count of {}: {}".format(asked_bigram,
    window_1_bigrams_without_pos_tags.count(asked_bigram)))

window_1_bigrams = collocations_dict[3]["bigrams"]
window_1_bigrams_without_pos_tags = []
for ((word_1, pos_tag_1),(word_2, pos_tag_2)) in
    window_1_bigrams:
    window_1_bigrams_without_pos_tags.append((word_1, word_2
        ))
asked_bigram = ("old", "friend")
print("Count of {}: {}".format(asked_bigram,
    window_1_bigrams_without_pos_tags.count(asked_bigram)))
```

```
#part 1.f

asked_bigram = ("mr.","skimpole")
result = asked_bigram in collocations_dict[1]["
    candidate_collocations"]
print("Is the bigram {} a candidate collocation: {}".format(
    asked_bigram, result))



#part 1.f
asked_bigram = ("large", "fortune")
result = asked_bigram in collocations_dict[3]["
    candidate_collocations"]
print("Is the bigram {} a candidate collocation: {}".format(
    asked_bigram, result))
```

```
### Part 2: Finding the Collocations
import pandas as pd
import numpy as np
from scipy.stats import binom
from IPython.display import display
```

```
# calculating t-score
def t_test_score(bigram, bigram_counts, word_counts, N):
    word1 = bigram[0]
    word2 = bigram[1]
    bigram_count = bigram_counts[bigram]
    word1_count = word_counts["word1"][word1]
    word2_count = word_counts["word2"][word2]
    # t_score = sample_mean - population_mean / (
        sample_variance / N)^0.5
    mle = bigram_count / N
    h0 = (word1_count/N) * (word2_count/N)
    sample_mean = mle
    sample_variance = mle # as p is small, p is approximated
         by mle
    t_score = (sample_mean - h0) / (sample_variance / N)
        **0.5
    return t_score
```

```python
# calculating chi-square score
def chi_square_score(bigram, bigram_counts, word_counts, N):
    word1 = bigram[0]
    word2 = bigram[1]
    bigram_count = bigram_counts[bigram]
    word1_count = word_counts["word1"][word1]
    word2_count = word_counts["word2"][word2]
    #observed matrix
    observed = np.array([[bigram_count, word2_count -
       bigram_count],
                            [word1_count - bigram_count, N - (
                               word1_count + word2_count -
                               bigram_count)]
                          ], dtype=np.float64)

    X2_num = N * (observed[0][0]*observed[1][1] - observed
       [0][1]*observed[1][0])**2
    X2_denum = (observed[0][0] + observed[0][1]) * (observed
       [0][0] + observed[1][0]) * (observed[0][1] + observed
       [1][1]) * (observed[1][0] + observed[1][1])
    X2 = X2_num / X2_denum
    return X2
```

```python
# calculating the likelihood ratio score
def likelihood_ratio_score(bigram, bigram_counts,
   word_counts, N):
    word1 = bigram[0]
    word2 = bigram[1]
    c12 = bigram_counts[bigram]
    c1 = word_counts["word1"][word1]
    c2 = word_counts["word2"][word2]
    p = c2 / N
    p1 = c12 / c1
    p2 = (c2 - c12) / (N - c1)
    threshold = 1e-100 # to not to calculate log(0),
       threshold is used
    #likelihood of h1(independence), using binomial
       distribution
    L1_1 = np.log(max(binom.pmf(c12, c1, p), threshold))
    L1_2 = np.log(max(binom.pmf(c2 - c12, N - c1, p),
       threshold))
    #likelihood of h2(dependence), using binomial
       distribution
    L2_1 = np.log(max(binom.pmf(c12, c1, p1), threshold))
    L2_2 = np.log(max(binom.pmf(c2 - c12, N - c1, p2),
       threshold))
    #likelihood ratio score
    LR = -2 * (L1_1 + L1_2 - L2_1 - L2_2)
    return LR
```

```
def dict_to_df(collocations_dict, window_size):
    bigram_counts = collocations_dict["bigram_counts"]
    word_counts = collocations_dict["word_counts"]
    candidate_collocations = collocations_dict["
       candidate_collocations"]

    data = []

    for bigram in candidate_collocations:
        w1, w2 = bigram
        c_w1w2 = bigram_counts.get(bigram, 0)
        c_w1 = word_counts["word1"].get(w1, 1)  # Avoid
           division by zero
        c_w2 = word_counts["word2"].get(w2, 1)
        t_score = t_test_score(bigram, bigram_counts,
           word_counts, N*window_size)
        chi_score = chi_square_score(bigram, bigram_counts,
           word_counts,  N*window_size)
        g_score = likelihood_ratio_score(bigram,
           bigram_counts, word_counts, N*window_size)

        # Append data as a row in the list
        data.append([w1+ ' '+ w2, t_score, chi_score,
           g_score, c_w1w2, c_w1, c_w2])

    # Create a DataFrame
    df = pd.DataFrame(data, columns=["w1w2",  "t-score", "
        chi-squared", "likelihood", "c(w1w2)", "c(w1)", "c(w2
        )"])

    return df
```

```
# create df for both window sizes
df_window_1 = dict_to_df(collocations_dict[1], 1)
df_window_3 = dict_to_df(collocations_dict[3], 3)
```

```
# window size: 1, sort by t-score
df_window_1_t_sorted = df_window_1.sort_values(by="t-score",
     ascending=False, ignore_index=True).reset_index(drop=
   True)
df_window_1_t_sorted.index += 1
print("Window size 1, sorted by t-score:")
display(df_window_1_t_sorted.head(20).drop(columns=["chi-
   squared", "likelihood"]))
```

```
# window size: 1, sort by chi-square-score
df_window_1_chi_sorted = df_window_1.sort_values(by="chi-
   squared", ascending=False, ignore_index=True).reset_index
   (drop=True)
df_window_1_chi_sorted.index += 1
print("Window size 1, sorted by chi-square score:")
display(df_window_1_chi_sorted.head(20).drop(columns=["t-
   score", "likelihood"]))
```

```
# window size: 1, sort by likelihood ratio score
df_window_1_g_sorted = df_window_1.sort_values(by="
   likelihood", ascending=False, ignore_index=True).
   reset_index(drop=True)
df_window_1_g_sorted.index += 1
print("Window size 1, sorted by likelihood ratio score:")
display(df_window_1_g_sorted.head(20).drop(columns=["t-score
   ", "chi-squared"]))
```

```
# window size: 3, sort by t-score
df_window_3_t_sorted = df_window_3.sort_values(by="t-score",
     ascending=False, ignore_index=True).reset_index(drop=
   True)
```

```
df_window_3_t_sorted.index += 1
print("Window size 3, sorted by t-score:")
display(df_window_3_t_sorted.head(20).drop(columns=["chi-
    squared", "likelihood"]))
```

```
# window size: 3, sort by chi-square-score
df_window_3_chi_sorted = df_window_3.sort_values(by="chi-
    squared", ascending=False, ignore_index=True).reset_index
    (drop=True)
df_window_3_chi_sorted.index += 1
print("Window size 3, sorted by chi-square score:")
display(df_window_3_chi_sorted.head(20).drop(columns=["t-
    score", "likelihood"]))
```

```
# window size: 3, sort by likelihood ratio score
df_window_3_g_sorted = df_window_3.sort_values(by="
    likelihood", ascending=False, ignore_index=True).
    reset_index(drop=True)
df_window_3_g_sorted.index += 1
print("Window size 3, sorted by likelihood ratio score:")
display(df_window_3_g_sorted.head(20).drop(columns=["t-score
    ", "chi-squared"]))
```

```
# print values for good wish with window size 1
good_wish = ("good", "wish")
good_wish_row = df_window_1[df_window_1["w1w2"] == "good
    wish"]
display(good_wish_row)



# print values for high spirit with window size 1
high_spirit = ("high", "spirit")
high_spirit_row = df_window_1[df_window_1["w1w2"] == "high
    spirit"]
display(high_spirit_row)
```