Proyect Structure

1. Data exploration and processing

2. Sexism detector:

    2.1 Dictionary-Based Sentiment Analysis to create sexism score

    2.2 Quicksort to organize data by sexism score

    2.3 Markov Chains for word predictions

    2.4 Algorithm for matrix multiplication (Strassen)

    2.5 Logistic regression with Strassen matrix multiplication and
    Gradient Descent

    2.6 Co-ocurrence tree to analyze words commonly used together
    in sexist tweets

Import Libraries

In [32]:
```
#Import the neccesary libraries
# fasttext: commonly used for natural language processing tasks
# io: for inout/output operations needed for interacting with data. Reading fr
# re: regular expressions are sequences of characters that define a search pat
# nltk: the "stopwords" module from the NLTK library provides a predefined lis
# PrettyTable: allows us to visualize the Markov Chain in a simple way

import fasttext
import io
import re
import nltk
nltk.download('stopwords')
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import random
import csv
import sys

#All of these were used for the logistic regression with matrix multiplication
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```
# #This library was ONLY used in step 2.5.1 (this is a bonus algorithm I inclu
# from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/mclevesluna/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:
```
stopwords = nltk.corpus.stopwords
```

1. Data exploration and processing

In [3]:
```
# Function to read CSV file with error handling and a custom delimiter
def read_csv_with_error_handling(file_path, delimiter=';'):
    data = []
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
        reader = csv.reader(file, delimiter=delimiter)
        for line_num, row in enumerate(reader, start=1):
            try:
                data.append(row)
            except csv.Error as e:
                print(f'Error at line {line_num}: {e}')
    return data

# Import training data with error handling and semicolon delimiter
df_tra = pd.DataFrame(read_csv_with_error_handling("../Project/EXIST_2021_Data
df_training = df_tra.sample(frac=0.1, random_state=42)
# Make sure that "source" and "task1" are read as strings
df_training[2] = df_training[2].astype(str)
df_training[5] = df_training[5].astype(str)

# Testing data with error handling and semicolon delimiter
df_tes = pd.DataFrame(read_csv_with_error_handling("../Project/EXIST_2021_Data
df_test = df_tes.sample(frac=0.1, random_state=42)
# Make sure that "source" and "task1" are read as strings
df_test[2] = df_test[2].astype(str)
df_test[5] = df_test[5].astype(str)
```

In [4]:
```
print(df_training.head())
```

```
              0     1        2   3  \
2908  EXIST2021  2935  twitter  en
2666  EXIST2021  2693  twitter  en
5809  EXIST2021  5887  twitter  es
5832  EXIST2021  5910  twitter  es
3710  EXIST2021  3760  twitter  es

                                                      4           5  \
2908  Ex-#Cuomo Aide: He '#Sexually #Harassed Me for...      sexist
2666         @Cannedbirds I dont hit women but probably      sexist
5809  @ldpsincomplejos Va saliendo todo a la luz, la...      sexist
5832  Esta Claudia no es más que una lagartona que t...      sexist
3710  @Toni0084 abortar no es desear la muerte, nadi...  non-sexist

                                 6
2908               sexual-violence
2666  misogyny-non-sexual-violence
5809  misogyny-non-sexual-violence
5832  misogyny-non-sexual-violence
3710                    non-sexist
```

In [5]:
```
print(df_test.head())
```

```
              0      1        2   3  \
2347  EXIST2021   9324  twitter  es
2399  EXIST2021   9376  twitter  es
1564  EXIST2021   8541  twitter  en
3989  EXIST2021  10966  twitter  es
3279  EXIST2021  10256  twitter  es

                                                      4           5  \
2347  @anluma99 @abulelrafas Mal que lo hubiera hech...  non-sexist
2399  Me explicaron que cuando los hombres abren las...      sexist
1564  @olamiposiabeni @mobolajinafisa1 @FaisalokoMor...  non-sexist
3989  Eu segurando o choro quando o Chris canta ,ÄúV...  non-sexist
3279  @maic00n__ Mitoooo #mgtow #gayscombolsonaro #lgbt  non-sexist

                           6
2347              non-sexist
2399  ideological-inequality
1564              non-sexist
3989              non-sexist
3279              non-sexist
```

For future reference:

2 = "source" 3 = "language" 4 = "text" 5 = "task1" 6 = "task2"

In [6]:
```
#Clean both spanish and english tweets (remove spaces, tags, links, make every

def clean_text(text, language):
    if language == "en":
        # keep only words
        remove_links = re.sub(r"(https?\://)\S+", "link", text)
        remove_tags = re.sub(r"(?:\@)\S+", "tag", text)
        letters_only_text = re.sub("[^a-zA-Z]", " ", remove_links)
        # convert to lower case and split
        words = letters_only_text.lower().split()
        # remove stopwords
        stopword_set = set(stopwords.words("english"))
```

```
        meaningful_words = [w for w in words if w not in stopword_set]
        # join the cleaned words in a list
        return " ".join(meaningful_words)

    #what do clean if its in spanish
    else:
        # keep only words
        remove_links = re.sub(r"(https?\://)\S+", "link", text)
        remove_tags = re.sub(r"(?:\@)\S+", "tag", text)
        letters_only_text = re.sub("[^abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLM
        # convert to lower case and split
        words = letters_only_text.lower().split()
        # remove stopwords
        stopword_set = set(stopwords.words("spanish"))
        meaningful_words = [w for w in words if w not in stopword_set]
        # join the cleaned words in a list
        return " ".join(meaningful_words)

# Create a new column for cleaned text in the training set bases off of curren
df_training['clean_text'] = df_training.apply(lambda row: clean_text(row[4], r

# Create a new column for cleaned text in the test set
df_test['clean_text'] = df_test.apply(lambda row: clean_text(row[4], row[3]), 

#Make sure "task1" has no trailing or leading spaces
df_training[5] = df_training[5].str.strip()
df_test[5] = df_test[5].str.strip()
```

2. Sexism detector

Sentiment analysis to calculate a sexism score

In [7]:
```
# Create dictionary of sexist words using the task1 in our training set
sexist_words = df_training[df_training[5] == 'sexist']['clean_text'].str.split
df_sexist_dict = set(sexist_words)
```

In [8]:
```
# Create function to analyze and score sexism based on the number of sexist wo
def sentiment_analysis(text, df_sexist_dict):
     # Split the cleaned tweet into words
    words = text.split()

    # Calculate the total sentiment score for the tweet
    total_score = sum([word in df_sexist_dict for word in words])

    # Define a threshold for classification (0 because we won't tolerate even 
    threshold = 0

    return total_score

#The complecity for this algorithm is: O(N + M * N), where N is the number of 
```

In [9]:
```
# Apply sentiment analysis to our training and test dataset and add as a colum
df_training["sexism_score"] = df_training["clean_text"].apply(lambda x: sentim
df_training.head()
```

```
df_test["sexism_score"] = df_test["clean_text"].apply(lambda x: sentiment_analy
df_test.head()
```

Out[9]:

	0	1	2	3	4	5	6	clean_text	s
2347	EXIST2021	9324	twitter	es	@anluma99 @abulelrafas Mal que lo hubiera hech...	non-sexist	non-sexist	anluma abulelrafas mal hecho miembro partido j...	
2399	EXIST2021	9376	twitter	es	Me explicaron que cuando los hombres abren las...	sexist	ideological-inequality	explicaron hombres abren piernas locomoci n p ...	
1564	EXIST2021	8541	twitter	en	@olamiposiabeni @mobolajinafisa1 @FaisalokoMor...	non-sexist	non-sexist	olamiposiabeni mobolajinafisa faisalokomori ab...	
3989	EXIST2021	10966	twitter	es	Eu segurando o choro quando o Chris canta ‚ÄúV...	non-sexist	non-sexist	eu segurando choro quando chris canta úvamo te...	
3279	EXIST2021	10256	twitter	es	@maic00n__ Mitoooo #mgtow #gayscombolsonaro #lgbt	non-sexist	non-sexist	maic n mitoooo mgtow gayscombolsonaro lgbt	

Quicksort to organize by sexism score

In [10]:

```
#Organize the data by the sexism score we created in our sentiment analysis alg

def randomized_quicksort(data, column='sexism_score'):
    if len(data) <= 1:
        return data

    # Randomly choose a pivot index
    pivot_index = np.random.randint(0, len(data))
    pivot = data[column].iloc[pivot_index]

    # Split the DataFrame
    less = data[data[column] < pivot]
    equal = data[data[column] == pivot]
    greater = data[data[column] > pivot]

    # Sort the split data
    return pd.concat([randomized_quicksort(less, column), equal, randomized_qui

# Sort the DataFrame by the 'sexism_score' column
sorted_dfTrain = randomized_quicksort(df_training, 'sexism_score')

# Display the sorted DataFrame
print(sorted_dfTrain)
```

```
             0     1        2   3  \
0    EXIST2021  4108  twitter  es
1    EXIST2021  1128  twitter  en
2    EXIST2021    95  twitter  en
3    EXIST2021   709  twitter  en
4    EXIST2021   406      gab  en
..         ...   ...      ...  ..
685  EXIST2021  1383  twitter  en
686  EXIST2021  6484  twitter  es
687  EXIST2021  2645  twitter  en
688  EXIST2021  1781  twitter  en
689  EXIST2021  2612  twitter  en

                                                     4           5  \
0    @soysi_tambien @gabrielboric Retaguardia Estrecha  non-sexist
1    this is my cockits harder than a rockhorny hou...  non-sexist
2       @holyquor @Answerforu2 #NotAllMen admit defeat  non-sexist
3                    @FaisalokoMori Bearded women nko  non-sexist
4    @jodecivante And I highly doubt you’re ugly be...  non-sexist
..                                                 ...         ...
685  these boys start dating one day &amp; expect t...      sexist
686  @ainhoaeus @BcnInsania @Eritacus @ilusocial @m...      sexist
687  @EXPELincels @beeonroids @shahjoffe @Ponderer_...  non-sexist
688  @NinjaSocialist @MgtowRadical @CrossBiddy @nat...  non-sexist
689  @CrossBiddy @NinjaSocialist @SR_Duncan @Shotgu...      sexist

                          6  \
0                non-sexist
1                non-sexist
2                non-sexist
3                non-sexist
4                non-sexist
..                      ...
685  stereotyping-dominance
686  stereotyping-dominance
687              non-sexist
688              non-sexist
689         sexual-violence

                                            clean_text  sexism_score
0      soysi tambien gabrielboric retaguardia estrecha             0
1    cockits harder rockhorny hours clockin bathroo...             0
2           holyquor answerforu notallmen admit defeat             0
3                      faisalokomori bearded women nko             1
4                  jodecivante highly doubt ugly bestie             1
..                                                 ...           ...
685  boys start dating one day amp expect partner l...            33
686  ainhoaeus bcninsania eritacus ilusocial mariam...            34
687  expelincels beeonroids shahjoffe ponderer purg...            56
688  ninjasocialist mgtowradical crossbiddy natspra...            57
689  crossbiddy ninjasocialist sr duncan shotgunrai...            86

[690 rows x 9 columns]
```

In [11]:
```
#The complexity of this quicksort algorithm is: O(n log n), where 'n' is the nu

#Because we are selecting hte pivot randomnly, the worst case scenario is very
```

Markov chain to predict future words from user

In [12]:
```
#We first need to tokenize our "clean_text" column to separate it into individu
tokenized_data = [clean_text(tweet, language) for tweet, language in zip(df_tra
```

In [13]:
```
# Build Markov Chain
def build_markov_chain(inputtext):
    chain = {}

    for tweet in inputtext:
        words = tweet.split()
        for i in range(len(words) - 1):
            current_word = words[i]
            next_word = words[i + 1]

            if current_word in chain:
                chain[current_word].append(next_word)
            else:
                chain[current_word] = [next_word]

    return chain

markov_chain = build_markov_chain(tokenized_data)

# Convert Markov Chain to DataFrame
df_markov_chain = pd.DataFrame(list(markov_chain.items()), columns=['Word', 'Ne

# Display the DataFrame
print(df_markov_chain)
```

```
                Word                           Next Words
0                 ex  [cuomo, reina, girlfriend, wife]
1              cuomo                            [aide]
2               aide                        [sexually]
3           sexually                        [harassed]
4           harassed             [years, theestallion]
...              ...                               ...
5773           lizzo                          [saying]
5774           loose                          [weight]
5775          weight                       [healthier]
5776       healthier                         [somehow]
5777  discriminating                             [fat]

[5778 rows x 2 columns]
```

In [14]:
```
#Now, also using our training data, we will calculate the transition probabili

probabilities = {}
def calculate_transition_probabilities(chain):

    for current_word, next_words in chain.items():
        total_next_words = len(next_words)
        probabilities[current_word] = {word: next_words.count(word) / total_ne

    return probabilities

transition_probabilities = calculate_transition_probabilities(markov_chain)

# Convert probabilities to DataFrame
df_transition_probabilities = pd.DataFrame(list(transition_probabilities.items
```

```
# Extract words and probabilities
words = df_transition_probabilities['Word'].tolist()
probabilities_matrix = df_transition_probabilities['Next Words'].apply(pd.Seri

# Assuming you have a list of words (cleaned_text)
cleaned_text = " ".join(df_training['clean_text'])
```

In [15]:
```
len(cleaned_text)
```

Out[15]:
```
80473
```

In [16]:
```
#Next we have to generate the future sequences
def generate_sequence(probabilities, seed, length=10):
    current_word = seed
    sequence = [current_word]

    for _ in range(length):
        if current_word in probabilities.keys():
            next_word = random.choices(list(probabilities[current_word].keys()
            sequence.append(next_word)
            current_word = next_word
        else:
            break

    return ' '.join(sequence)

seed_word = "woman"
generated_sequence = generate_sequence(probabilities, seed=seed_word, length=1
print(generated_sequence)
```

```
woman bet would love white women sammy club dejan pueden ser jóvenes imprudent
es creemos sabemos todas
```

In [17]:
```
#Complexity of the full Markov chain:  O(N * M).The where N is the number of t
```

Strassen's Algorithm for Matrix Multiplication

In [18]:
```
def split_matrix(matrix):
    # Check if the matrix has only one dimension
    if matrix.ndim == 1:
        # If it's a 1D array, convert it to a 2D column vector
        matrix = matrix.reshape((-1, 1))

    # the matrixes must be split into quadrants first
    row, col = matrix.shape
    row2, col2 = row // 2, col // 2

    upper_left = matrix[:row2, :col2]
    upper_right = matrix[:row2, col2:]
    lower_left = matrix[row2:, :col2]
    lower_right = matrix[row2:, col2:]

    return upper_left, upper_right, lower_left, lower_right

def strassen_multiply(A, B,threshold=50000):

    # Base case: switch to a more efficient algorithm (e.g., NumPy)
    if A.shape[0] <= threshold:
        return np.dot(A, B)
```

```
    # Split matrices into four quadrants
    a, b, c, d = split_matrix(A)
    e, f, g, h = split_matrix(B)

    # Recursive steps for Strassen's algorithm
    p1 = strassen_multiply(a, f - h)
    p2 = strassen_multiply(a + b, h)
    p3 = strassen_multiply(c + d, e)
    p4 = strassen_multiply(d, g - e)
    p5 = strassen_multiply(a + d, e + h)
    p6 = strassen_multiply(b - d, g + h)
    p7 = strassen_multiply(a - c, e + f)

    # Compute the quadrants of the result matrix
    upper_left = p5 + p4 - p2 + p6
    upper_right = p1 + p2
    lower_left = p3 + p4
    lower_right = p1 + p5 - p3 - p7

    # Combine the quadrants to get the result matrix
    result = np.vstack((np.hstack((upper_left, upper_right)),
                        np.hstack((lower_left, lower_right))))

    return result
```

In [19]:
```
#Create some quick test matrices to ensure our Strassen's Algorithm for Matrix

# Define two matrices A and B
A = np.array([[4, 3], [5, 2]])
B = np.array([[5, 1], [5, 6]])

# Multiply matrices using Strassen's algorithm
result = strassen_multiply(A,B)

print("A_matrix:")
print(A)
print("B")
print(B)
print("\nResult of Matrix Multiplication:")
print(result)
```

```
A_matrix:
[[4 3]
 [5 2]]
B
[[5 1]
 [5 6]]

Result of Matrix Multiplication:
[[35 22]
 [35 17]]
```

In [20]:
```
#Overall time complexity of the algorithm: O(n**log(2)7)=O(n**2.81), where n i
```

Logistic Regression with Matrix Multiplication (Strassen) and Gradient Descent

In order to use two algorithms in one task, we will be doing a manual logistic regression with matrix multiplication. Our matrix multiplication will be conducted using the Strassen's

Algorithm defined in the previous function.

In [21]:
```
df_training[2]
```

Out[21]:
```
2908    twitter
2666    twitter
5809    twitter
5832    twitter
3710    twitter
         ...
3257    twitter
1599    twitter
4536    twitter
1009    twitter
734         gab
Name: 2, Length: 690, dtype: object
```

In [22]:
```
#Convert binary categoric variables (sources and language) into numeric ones
df_training['numeric_language']= df_training[3].apply(lambda x: 1 if x == 'en'
df_test['numeric_language']=df_test[3].apply(lambda x: 1 if x == 'en' else (2 

df_training['numeric_source']= df_training[2].apply(lambda x: 1 if x == 'twitte
df_test['numeric_source']=df_test[2].apply(lambda x: 1 if x == 'twitter' else 

df_test['sexism_score'] = pd.to_numeric(df_test['sexism_score'], errors='coerce
df_training['sexism_score'] = pd.to_numeric(df_training['sexism_score'], errors
df_test['numeric_language'] = pd.to_numeric(df_test['numeric_language'], errors
df_training['numeric_language'] = pd.to_numeric(df_training['numeric_language']
df_test['numeric_source'] = pd.to_numeric(df_test['numeric_source'], errors='c
df_training['numeric_source'] = pd.to_numeric(df_training['numeric_source'], e

df_training[5]= df_training[5].apply(lambda x: 1 if x == 'sexist' else (0 if x 
df_test[5]=df_test[5].apply(lambda x: 1 if x == 'sexist' else (0 if x == 'non-s

X = df_training[['numeric_source','numeric_language', 'sexism_score']]

# Convert our sexism output into a NumPy array for matrix multiplication
y = df_training[5].values
# Convert our variables array for matrix multiplication
X_matrix = X.values

#Gettting my aprameters ready
# Initialize and transposing theta
theta = np.zeros(X_matrix.shape[1])
thetaT = np.transpose(theta)
```

In [23]:
```
# Define the sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Define the cost function
def cost_function(X, y, theta):
    m = len(y)
    h = sigmoid(strassen_multiply(X, theta,threshold=50000))

    cost = y * np.log(h) + (1 - y) * np.log(1-h)
    cost = -np.sum(cost)/m

    return cost
```

```
# Define the gradient descent function
def gradient_descent(X, y, theta, learning_rate, epochs):
    m = len(y)
    for epoch in range(epochs):

        h = sigmoid(strassen_multiply(X, theta,threshold=50000))

        gradient = (1/m) * strassen_multiply(X.T,(h - y),threshold=50000)

        theta = theta - learning_rate * gradient

        cost = cost_function(X, y, theta)

        if epoch % 2000 == 0:
            print(f'Epoch {epoch}, Cost: {cost}')

    return theta
```

In [24]:
```
# Train the model
learning_rate = 0.01
epochs = 50000

def normalize(X):
    return (X-np.min(X))/(np.max(X)-np.min(X)+1e-6)

X = np.float16(X_matrix)
y = df_training[5].values
y = np.float16(y[:,np.newaxis])
for i in range(2):
    print(np.min(X[:,i]), np.max(X[:,i]))
    X[:,i] = normalize(X[:,i])
X = np.hstack((X, np.ones((X.shape[0], 1), dtype=X.dtype)))
y = normalize(y)
theta = np.zeros(X.shape[1])
theta = theta[:,np.newaxis]

theta_final = gradient_descent(X, y, theta, learning_rate, epochs)
theta_final
```

```
1.0 2.0
1.0 2.0
Epoch 0, Cost: 0.6584125201751524
Epoch 2000, Cost: 0.5609166082909162
Epoch 4000, Cost: 0.5519517053232531
Epoch 6000, Cost: 0.5494692204507458
Epoch 8000, Cost: 0.5486331269132615
Epoch 10000, Cost: 0.548323119111504
Epoch 12000, Cost: 0.548195805362609
Epoch 14000, Cost: 0.5481367127968545
Epoch 16000, Cost: 0.5481054932049796
Epoch 18000, Cost: 0.548087041764228
Epoch 20000, Cost: 0.5480752299946972
Epoch 22000, Cost: 0.5480672895848695
Epoch 24000, Cost: 0.5480618046599061
Epoch 26000, Cost: 0.5480579614902177
Epoch 28000, Cost: 0.548055249080892
Epoch 30000, Cost: 0.5480533277982507
Epoch 32000, Cost: 0.5480519644677279
Epoch 34000, Cost: 0.5480509962167659
Epoch 36000, Cost: 0.5480503082689888
Epoch 38000, Cost: 0.5480498193829668
Epoch 40000, Cost: 0.5480494719286608
Epoch 42000, Cost: 0.5480492249818979
Epoch 44000, Cost: 0.548049049466964
Epoch 46000, Cost: 0.5480489247213733
Epoch 48000, Cost: 0.5480488360599325
```

Out[24]:
```
array([[ 0.1776294 ],
       [ 0.26705672],
       [ 0.17877411],
       [-1.98711691]])
```

In [25]:
```
#Define test matrices
y_test = df_test[5]
y_test = np.array(y_test)
X_test = df_test[['numeric_source','numeric_language','sexism_score']]
```

In [26]:
```
# Make predictions on test set

predictions = np.array(sigmoid(np.dot(X, theta_final)))
y_test = df_test[5]

# print(predictions[:,0])
print(cost_function(X, y, theta_final))
```

```
0.5480487730716307
```

In [27]:
```
predictions
```

Out[27]:
```
array([[0.36425949],
       [0.25100747],
       [0.94921657],
       [0.26796661],
       [0.6862784 ],
       [0.60473387],
       [0.42803636],
       [0.92893993],
       [0.98736114],
       [0.4065732 ],
       [0.16388946],
       [0.42803636],
       [0.20383235],
       [0.30445136],
       [0.30445136],
       [0.70542544],
       [0.23438079],
       [0.89327964],
       [0.28608555],
       [0.62614658],
       [0.34357218],
       [0.77396156],
       [0.21891158],
       [0.15186341],
       [0.95340552],
       [0.93987339],
       [0.98202635],
       [0.53946088],
       [0.32394556],
       [0.9161956 ],
       [0.90140822],
       [0.99967268],
       [0.4065732 ],
       [0.51691378],
       [0.23438079],
       [0.56130378],
       [0.26796661],
       [0.4065732 ],
       [0.75773942],
       [0.64657166],
       [0.51691378],
       [0.92288707],
       [0.26796661],
       [0.58344862],
       [0.830386  ],
       [0.4065732 ],
       [0.49484977],
       [0.34357218],
       [0.86476246],
       [0.81725716],
       [0.66696607],
       [0.70542544],
       [0.26796661],
       [0.78903189],
       [0.16388946],
       [0.14083853],
       [0.74116702],
       [0.51691378],
       [0.58344862],
       [0.56130378],
```

```
[0.86476246],
[0.81725716],
[0.90140822],
[0.93468636],
[0.53946088],
[0.47225524],
[0.56102188],
[0.25100747],
[0.45032052],
[0.47225524],
[0.92288707],
[0.25100747],
[0.30445136],
[0.9161956 ],
[0.75773942],
[0.92893993],
[0.26796661],
[0.18987931],
[0.64657166],
[0.95717404],
[0.6862784 ],
[0.94921657],
[0.23438079],
[0.47225524],
[0.87500038],
[0.64657166],
[0.47225524],
[0.21891158],
[0.89327964],
[0.34357218],
[0.4065732 ],
[0.18987931],
[0.32394556],
[0.64657166],
[0.77396156],
[0.85410158],
[0.56130378],
[0.20383235],
[0.64657166],
[0.30445136],
[0.6862784 ],
[0.32369492],
[0.34357218],
[0.94921657],
[0.18987931],
[0.72343234],
[0.53946088],
[0.53946088],
[0.45032052],
[0.74116702],
[0.56130378],
[0.42803636],
[0.70542544],
[0.58344862],
[0.84246016],
[0.85410158],
[0.42803636],
[0.62614658],
[0.74116702],
[0.51691378],
```

```
[0.38493786],
[0.93468636],
[0.16388946],
[0.30445136],
[0.95717404],
[0.38493786],
[0.38466687],
[0.53946088],
[0.23438079],
[0.38493786],
[0.23438079],
[0.93468636],
[0.16388946],
[0.47225524],
[0.16388946],
[0.90906876],
[0.9161956 ],
[0.87500038],
[0.42803636],
[0.21891158],
[0.36425949],
[0.28608555],
[0.92893993],
[0.4065732 ],
[0.77396156],
[0.53946088],
[0.45032052],
[0.36425949],
[0.32394556],
[0.36425949],
[0.25079232],
[0.75773942],
[0.16373266],
[0.90916334],
[0.92893993],
[0.62614658],
[0.1763484 ],
[0.56130378],
[0.47225524],
[0.62614658],
[0.30445136],
[0.38493786],
[0.28608555],
[0.42803636],
[0.66696607],
[0.830386  ],
[0.28608555],
[0.4065732 ],
[0.42803636],
[0.36425949],
[0.36425949],
[0.45032052],
[0.25100747],
[0.56130378],
[0.70542544],
[0.14083853],
[0.70542544],
[0.62614658],
[0.36425949],
[0.16388946],
```

```
       [0.72343234],
       [0.26796661],
       [0.51662793],
       [0.40629704],
       [0.4065732 ],
       [0.4065732 ],
       [0.26796661],
       [0.9766267 ],
       [0.14083853],
       [0.95717404],
       [0.72343234],
       [0.28608555],
       [0.66671176],
       [0.58344862],
       [0.85410158],
       [0.6862784 ],
       [0.38493786],
       [0.16388946],
       [0.30445136],
       [0.830386  ],
       [0.18987931],
       [0.28608555],
       [0.14083853],
       [0.90916334],
       [0.92288707],
       [0.90916334],
       [0.38493786],
       [0.34357218],
       [0.51691378],
       [0.64657166],
       [0.47225524],
       [0.90916334],
       [0.51662793],
       [0.42803636],
       [0.30445136],
       [0.47225524],
       [0.53946088],
       [0.53946088],
       [0.4065732 ],
       [0.45032052],
       [0.18987931],
       [0.58344862],
       [0.60473387],
       [0.23438079],
       [0.30445136],
       [0.62614658],
       [0.64657166],
       [0.30445136],
       [0.20383235],
       [0.89327964],
       [0.34357218],
       [0.58344862],
       [0.62614658],
       [0.66671176],
       [0.94921657],
       [0.12056221],
       [0.97449794],
       [0.90140822],
       [0.18987931],
       [0.14083853],
```

```
[0.51691378],
[0.90140822],
[0.21871591],
[0.42803636],
[0.56130378],
[0.64657166],
[0.21891158],
[0.93468636],
[0.64657166],
[0.9161956 ],
[0.21891158],
[0.53946088],
[0.90916334],
[0.4065732 ],
[0.26796661],
[0.66671176],
[0.75773942],
[0.42803636],
[0.72343234],
[0.58344862],
[0.42803636],
[0.28608555],
[0.77396156],
[0.58344862],
[0.90916334],
[0.77396156],
[0.26796661],
[0.51691378],
[0.36425949],
[0.12056221],
[0.1763484 ],
[0.96393201],
[0.4065732 ],
[0.25100747],
[0.16388946],
[0.28608555],
[0.20383235],
[0.47225524],
[0.42803636],
[0.66696607],
[0.36425949],
[0.32394556],
[0.20383235],
[0.28608555],
[0.21891158],
[0.1763484 ],
[0.49484977],
[0.93987339],
[0.34357218],
[0.90140822],
[0.93987339],
[0.56130378],
[0.60473387],
[0.1763484 ],
[0.30445136],
[0.20383235],
[0.92288707],
[0.90916334],
[0.25100747],
[0.74094736],
```

```
       [0.66696607],
       [0.47225524],
       [0.90916334],
       [0.88434097],
       [0.6862784 ],
       [0.92893993],
       [0.64657166],
       [0.95340552],
       [0.34357218],
       [0.23438079],
       [0.1763484 ],
       [0.30420901],
       [0.92893993],
       [0.18987931],
       [0.25100747],
       [0.47225524],
       [0.18987931],
       [0.53946088],
       [0.72343234],
       [0.62614658],
       [0.95717404],
       [0.25100747],
       [0.28608555],
       [0.30445136],
       [0.56130378],
       [0.90916334],
       [0.36425949],
       [0.47225524],
       [0.16388946],
       [0.38493786],
       [0.6862784 ],
       [0.38493786],
       [0.4065732 ],
       [0.34357218],
       [0.23438079],
       [0.4065732 ],
       [0.90140822],
       [0.38493786],
       [0.38493786],
       [0.4065732 ],
       [0.98039996],
       [0.60473387],
       [0.56130378],
       [0.28585182],
       [0.14083853],
       [0.20383235],
       [0.32394556],
       [0.1763484 ],
       [0.80370147],
       [0.47225524],
       [0.28608555],
       [0.62614658],
       [0.28608555],
       [0.34357218],
       [0.18987931],
       [0.49456362],
       [0.16388946],
       [0.78903189],
       [0.51691378],
       [0.38493786],
```

```
       [0.21891158],
       [0.51662793],
       [0.53946088],
       [0.30445136],
       [0.36425949],
       [0.830386  ],
       [0.21891158],
       [0.25100747],
       [0.18987931],
       [0.32394556],
       [0.84246016],
       [0.84246016],
       [0.53917647],
       [0.96965737],
       [0.30445136],
       [0.32394556],
       [0.30445136],
       [0.51691378],
       [0.45032052],
       [0.34357218],
       [0.74116702],
       [0.14083853],
       [0.49484977],
       [0.62614658],
       [0.28608555],
       [0.92288707],
       [0.51691378],
       [0.49484977],
       [0.88434097],
       [0.14083853],
       [0.38493786],
       [0.30445136],
       [0.26796661],
       [0.32394556],
       [0.16388946],
       [0.18987931],
       [0.42803636],
       [0.90140822],
       [0.28608555],
       [0.42803636],
       [0.88434097],
       [0.28608555],
       [0.51691378],
       [0.4065732 ],
       [0.42803636],
       [0.62614658],
       [0.47225524],
       [0.51691378],
       [0.26796661],
       [0.94921657],
       [0.74116702],
       [0.42803636],
       [0.23438079],
       [0.20383235],
       [0.18987931],
       [0.70542544],
       [0.93987339],
       [0.21871591],
       [0.23438079],
       [0.90916334],
```

```
[0.36425949],
[0.4065732 ],
[0.90140822],
[0.42803636],
[0.66696607],
[0.51691378],
[0.30445136],
[0.32394556],
[0.74116702],
[0.51691378],
[0.4065732 ],
[0.77396156],
[0.85410158],
[0.20383235],
[0.25079232],
[0.62614658],
[0.36425949],
[0.21891158],
[0.25100747],
[0.53946088],
[0.98202635],
[0.56130378],
[0.36425949],
[0.70542544],
[0.62614658],
[0.42803636],
[0.32394556],
[0.36425949],
[0.42803636],
[0.56130378],
[0.93468636],
[0.75773942],
[0.26796661],
[0.25079232],
[0.62614658],
[0.4065732 ],
[0.86476246],
[0.64657166],
[0.58344862],
[0.75773942],
[0.64657166],
[0.96393201],
[0.62614658],
[0.53946088],
[0.20383235],
[0.36425949],
[0.9669494 ],
[0.36425949],
[0.42803636],
[0.26796661],
[0.23438079],
[0.74116702],
[0.28608555],
[0.1763484 ],
[0.78903189],
[0.88434097],
[0.21891158],
[0.87500038],
[0.47225524],
[0.42803636],
```

```
       [0.42803636],
       [0.38493786],
       [0.21891158],
       [0.14083853],
       [0.38493786],
       [0.38493786],
       [0.95340552],
       [0.88434097],
       [0.14083853],
       [0.14083853],
       [0.81725716],
       [0.830386  ],
       [0.72343234],
       [0.56130378],
       [0.53917647],
       [0.23438079],
       [0.20383235],
       [0.86476246],
       [0.90140822],
       [0.18987931],
       [0.32394556],
       [0.30445136],
       [0.62614658],
       [0.34357218],
       [0.53946088],
       [0.87500038],
       [0.16388946],
       [0.84246016],
       [0.32394556],
       [0.36425949],
       [0.66696607],
       [0.23438079],
       [0.92893993],
       [0.47225524],
       [0.47225524],
       [0.99972625],
       [0.89327964],
       [0.53946088],
       [0.16388946],
       [0.23438079],
       [0.25079232],
       [0.25100747],
       [0.20383235],
       [0.45032052],
       [0.16388946],
       [0.28608555],
       [0.42803636],
       [0.83022471],
       [0.9447882 ],
       [0.16388946],
       [0.36425949],
       [0.26796661],
       [0.84246016],
       [0.42803636],
       [0.62614658],
       [0.42803636],
       [0.51691378],
       [0.93468636],
       [0.53946088],
       [0.34357218],
```

```
       [0.66696607],
       [0.28608555],
       [0.62614658],
       [0.45032052],
       [0.32394556],
       [0.84246016],
       [0.72343234],
       [0.16388946],
       [0.84246016],
       [0.38493786],
       [0.92288707],
       [0.42803636],
       [0.34357218],
       [0.58344862],
       [0.78903189],
       [0.38493786],
       [0.62614658],
       [0.26796661],
       [0.30445136],
       [0.51691378],
       [0.45032052],
       [0.70542544],
       [0.34357218],
       [0.92288707],
       [0.93987339],
       [0.58344862],
       [0.6862784 ],
       [0.38493786],
       [0.89327964],
       [0.32369492],
       [0.45032052],
       [0.80370147],
       [0.64657166],
       [0.85410158],
       [0.53946088],
       [0.25100747],
       [0.47225524],
       [0.30445136],
       [0.81725716],
       [0.4065732 ],
       [0.88434097],
       [0.85410158],
       [0.42803636],
       [0.60473387],
       [0.49484977],
       [0.81725716],
       [0.26796661],
       [0.26796661],
       [0.93468636],
       [0.38493786],
       [0.66696607],
       [0.1763484 ],
       [0.56130378],
       [0.830386  ],
       [0.74116702],
       [0.96965737],
       [0.47225524],
       [0.34357218],
       [0.95717404],
       [0.42803636],
```

```
[0.66696607],
[0.36425949],
[0.26796661],
[0.23438079],
[0.45032052],
[0.25100747],
[0.25100747],
[0.64657166],
[0.95717404],
[0.51691378],
[0.30445136],
[0.62614658],
[0.9161956 ],
[0.42803636],
[0.89327964],
[0.96069056],
[0.53946088],
[0.99999847],
[0.66696607],
[0.1763484 ],
[0.75773942],
[0.26796661],
[0.26796661],
[0.78903189],
[0.45032052],
[0.32394556],
[0.26796661],
[0.92288707],
[0.42803636],
[0.60473387],
[0.16388946],
[0.34357218],
[0.9722096 ],
[0.30445136],
[0.51691378],
[0.70542544],
[0.34357218],
[0.21891158],
[0.97449794],
[0.32394556],
[0.93980867],
[0.32394556],
[0.58344862],
[0.53946088],
[0.51691378],
[0.90140822],
[0.28608555],
[0.20383235],
[0.96393201],
[0.25100747],
[0.87500038],
[0.18987931],
[0.4065732 ],
[0.72343234],
[0.75773942],
[0.38493786],
[0.26796661],
[0.30445136],
[0.32394556],
[0.34357218],
```

```
       [0.30445136],
       [0.77396156],
       [0.66696607],
       [0.20383235],
       [0.34357218],
       [0.83022471],
       [0.25100747],
       [0.16388946],
       [0.9766267 ],
       [0.28608555],
       [0.30445136],
       [0.42803636],
       [0.40629704],
       [0.28608555],
       [0.23438079],
       [0.36425949],
       [0.34357218],
       [0.58344862],
       [0.75773942],
       [0.4065732 ],
       [0.18987931],
       [0.90916334],
       [0.58317039],
       [0.32394556],
       [0.38493786],
       [0.58344862],
       [0.97665281],
       [0.38493786],
       [0.18987931],
       [0.49456362]])
```

In [28]:
```
#The complexity of this algorithm depends mostly on the Gradient Descent and th
#Complexity: O(Matrix Multiplication+Gradient Descent)=O(n**2.81)+k·n·(d+1))
#Where k is the number of epochs, n is the number of instances, and d is the nu
```

Creating a co-ocurrences tree

In [29]:
```
# Remember, this is the sexist word dictionary we created in our sentiment ana
# Remember, we had tokenized our tweet data to create our Markov Chains: token

#Convert our df_sexist_dict into list
sexist_words = list(df_sexist_dict)
```

In [30]:
```
# Set a threshold for the sexism score
threshold = 11  # This corresponds to the mean sexism score in our test data

# Filter tweets based on the threshold
high_sexism_tweets = df_test[df_test['sexism_score'] > threshold]

# Convert the 'text' column to strings
high_sexism_tweets['tokenized_text'] = high_sexism_tweets[4].astype(str).apply

# Display the tokenized text
print(high_sexism_tweets[[4, 'tokenized_text']])

# Tokenize the text into words for high sexism tweets
tokenized_high_sexism_tweets = [text.split() for text in high_sexism_tweets['te

# Create a co-occurrence matrix considering only sexist words in high sexism tw
```

```
co_occurrence_matrix = {}
for words in tokenized_high_sexism_tweets:
    for i, word_i in enumerate(words):
        for j, word_j in enumerate(words):
            if i != j and (word_i in sexist_words or word_j in sexist_words):
                key = (word_i, word_j)
                co_occurrence_matrix[key] = co_occurrence_matrix.get(key, 0) +

# Print the 50 pairs with the highest weight (most common word combinations in
top_edges = sorted(co_occurrence_matrix.items(), key=lambda x: x[1], reverse=Tr

for edge, weight in top_edges:
    print(f"{edge[0]} -- {edge[1]}: {weight}")


#You can activate this part to visualize the network, but it will print out a l

# for edge, weight in co_occurrence_matrix.items():
#     print(f"{edge[0]} -- {edge[1]}: {weight}")

# The warning that appears is NOT critical or relevant so we will ignore it
```

```
                                                      4  \
4291  Al guionista de Superl√≥pez no le da la puta g...
1886  @xlizagx I will take the handles hun.. serious...
3364  Michelle Bachelet reconoce violaci√≥n de derec...
4368  @elmundoes A Pablo es que ya no le hacen caso ...
4347  @vania_vargas Concuerdo... Mi hija y el resto ...
...                                                 ...
1904  Best explanation I,Äôve seen of whole Google s...
1338  @HayliNic Grown ass woman, playing video games...
903   @CatfishKristen6 wields a psychosexual power o...
2252  @Berro_con_limon @MarioPscherer As√≠ son los m...
893   I hate when bitches say they gone beat my ass ...


                                          tokenized_text
4291  guionista superl pez da puta gana rodar blas l...
1886  xlizagx i will take the handles hun seriously ...
3364  michelle bachelet reconoce violaci n derechos ...
4368  elmundoes pablo hacen caso consejos soltar ton...
4347  vania vargas concuerdo hija resto mujeres debe...
...                                                 ...
1904  best explanation i ve seen of whole google sag...
1338  haylinic grown ass woman playing video games t...
903   catfishkristen wields psychosexual power over ...
2252  berro limon mariopscherer as machistas mujer c...
893   i hate when bitches say they gone beat my ass ...

[99 rows x 2 columns]
```

```
/var/folders/cc/24m71qvx7n14q697ws0scfb40000gn/T/ipykernel_6557/1946314466.py:
8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  high_sexism_tweets['tokenized_text'] = high_sexism_tweets[4].astype(str).app
ly(clean_text, language=2)
```

```
to -- women: 62
women -- to: 62
t -- i: 60
i -- t: 60
to -- s: 49
s -- to: 49
the -- women: 43
women -- the: 43
s -- s: 42
the -- in: 42
in -- the: 42
i -- hate: 40
hate -- i: 40
i -- in: 37
the -- t: 37
in -- i: 37
t -- the: 37
i -- women: 36
men -- women: 36
women -- i: 36
women -- men: 36
s -- the: 36
the -- s: 36
to -- men: 36
men -- to: 36
like -- i: 35
i -- like: 35
is -- s: 35
s -- is: 35
you -- t: 34
t -- you: 34
i -- s: 34
is -- women: 34
women -- is: 34
s -- i: 34
t -- it: 33
it -- t: 33
i -- at: 33
at -- i: 33
of -- women: 33
women -- of: 33
to -- t: 32
to -- in: 32
t -- to: 32
in -- to: 32
like -- to: 32
to -- like: 32
men -- men: 32
i -- love: 32
love -- i: 32
```

In [ ]:

For a later time, we will finish constructing the following:

Predicting wether the string of words generated by our Markov Chain will be sexist according to our logistic regression**

1. Preprocess the Generated String: Tokenize the generated string into words.

2. Apply any necessary cleaning or preprocessing steps that you used during the training of your logistic regression model.
3. Feature Extraction: Extract the same features from the generated string that were used as input features during the training of your logistic regression model. This may include word frequencies, presence of specific words, or any other relevant features.
4. Use Logistic Regression Model:Input the extracted features into your trained logistic regression model to obtain a prediction.

In [31]:
```
# # Preprocess the string generated by the Markov Chain
# tokenized_text = generated_sequence.split()
# processed_generated_sequence = " ".join(tokenized_text)

# # Convert X_matrix to a list of strings
# X_texts = [" ".join(map(str, row)) for row in X_matrix]

# # Fit and transform the processed text
# tfidf_vectorizer.fit(X_texts)
# generated_features = tfidf_vectorizer.transform([processed_generated_sequence

# # Add a constant term for bias to the features
# generated_features_with_bias = np.hstack([generated_features.toarray(), np.o

# # Assuming you have three features (numeric_source, numeric_language, sexism_
# num_features = 3

# # Ensure that the number of features matches the size of theta_final
# assert generated_features_with_bias.shape[1] == num_features + 1, "Number of

# # Now, make predictions using the logistic regression model
# generated_predictions = sigmoid(np.dot(generated_features_with_bias, theta_f

# # Print the predictions
# print("Generated Predictions:", generated_predictions)

# # Print the feature names (words)
# print("Feature names:", tfidf_vectorizer.get_feature_names_out())
# # Print the TF-IDF matrix
# print("TF-IDF matrix:\n", generated_features.toarray())

# Make predictions on the TF-IDF features
#predictions = predictions

# Print the predictions
#print("Predictions:", predictions)
```