In [58]:
```
import nltk
import pandas as pd
nltk.download('popular', quiet=True)
import demoji
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.util import bigrams
from nltk import FreqDist
import spacy
import string
nltk.download("wordnet", quiet=True)
nltk.download("stopwords", quiet=True)
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import pos_tag
from nltk.corpus import wordnet
from collections import defaultdict
import pandas as pd
nlp = spacy.load("en_core_web_sm")
```

In [59]:
```
import os
import glob
def collect_data():
    text_file_pattern = "*.txt"  # You can adjust the pattern to match your file extensions
    text_files = glob.glob(os.path.join("../nhs/content", text_file_pattern))
    data = {}
    for file_path in text_files:
        with open(file_path, 'r', encoding='utf-8') as file:
            file_name = os.path.basename(file_path)
            file_content = file.read()
            data[file_name] = file_content
    return data
```

In [60]:
```
corpus = collect_data()
text = ""
for data in corpus:
    text += " " + data
```

In [61]:
```
def remove_punctuation(text):
    # Create a translation table to remove punctuation
    translator = str.maketrans('', '', string.punctuation)
    # Use translate method to remove punctuation
    cleaned_text = text.translate(translator)
    return cleaned_text

def remove_stop_words(text):
    nltk_stopwords = stopwords.words('english')
    spacy_stopwords = nlp.Defaults.stop_words

    stop_words = (*nltk_stopwords, *spacy_stopwords, "NHStxt")

    tokens = word_tokenize(text)
    tokens = [token for token in tokens if token not in stop_words]
    return " ".join(tokens)

def apply_lemmitization(text):
    tag_map = defaultdict(lambda : wordnet.NOUN)
    tag_map['V'] = wordnet.VERB
    tag_map['A'] = wordnet.ADJ
    tag_map['R'] = wordnet.ADJ

    lemmitizer = WordNetLemmatizer()
    lemmitized_result = ""
    tokens = word_tokenize(text)
    for token, tag in pos_tag(tokens):
        lemma = lemmitizer.lemmatize(token, tag_map[tag[0]])
        lemmitized_result = lemmitized_result + " " + lemma
    return lemmitized_result

def remove_emoji_and_smart_quotes(text):
    # replacing emojis with description
    text = demoji.replace_with_desc(text)
    #Removing smart quotes
    return text.replace("“", "\"").replace("”","\"")
```

1. Perform necessary data preprocessing, e.g. removing punctuation and stop words, stemming, lemmatizing. You may use the outputs from previous weekly assignments. (10 points)

In [62]:
```
def data_preprocessing(text):
    text = remove_emoji_and_smart_quotes(text)
    text = remove_punctuation(text)
    text = remove_stop_words(text)
    text = apply_lemmitization(text)
```

```
    return text

def apply_data_preprocessing_to_corpus(corpus):
    new_corpus = {}
    for idx, key in enumerate(corpus.keys()):
        new_corpus[key] = data_preprocessing(corpus[key])
        print(f"idx: {idx}")
    return new_corpus
```

In [63]:
```
processed_text = data_preprocessing(text)
with open('week7_1.txt', 'w') as file:
    file.write(f'{processed_text}')
```

2. Count BoW on pre-processed data. (10 points)

In [64]:
```
def get_document_features(word_features, text):
    features = {}
    tokens = word_tokenize(text)
    for word in word_features:
        features[word] = tokens.count(word)
    return features

def get_corpus_features(corpus, text):
    text = text.lower()
    words = word_tokenize(text)
    word_features = list(FreqDist(words).most_common(1000))
    word_features = [i for i,j in word_features]
    dataframe_data = []
    for key in corpus.keys():
        features = get_document_features(word_features, corpus[key])
        features['title'] = key
        dataframe_data.append(features)
    return dataframe_data
```

In [65]:
```
df_data = get_corpus_features(corpus, processed_text)
df = pd.DataFrame(df_data)
df
```

Out[65]:

	syndrome	disease	pain	cancer	disorder	test	blood	eye	child	foot	...	bedbugs	versicolor	pilonidal	system	labyrinthitis	neuritis	tendon	clinical
0	0	0	0	0	1	0	0	0	7	0	...	0	0	0	0	0	0	0	0
1	0	0	1	10	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	0	2	0	0	0	0	6	1	0	0	...	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	2	0	0	1	1	0	1	0	...	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
973	0	0	3	0	0	1	1	0	1	1	...	0	0	0	1	0	0	0	0
974	0	0	0	0	0	0	0	5	0	0	...	0	0	0	0	0	0	0	0
975	0	16	0	0	0	1	1	0	12	0	...	0	0	0	0	0	0	0	1
976	11	2	3	0	1	0	2	0	2	0	...	0	0	0	2	0	0	0	0
977	0	2	0	0	0	0	15	0	0	0	...	0	0	0	0	0	0	0	0

978 rows × 1001 columns

3. Compute TF-IDF vectors on pre-processed data. (20 points)

In [66]:
```
processed_corpus = apply_data_preprocessing_to_corpus(corpus)
```

```
idx: 0
idx: 1
idx: 2
idx: 3
idx: 4
idx: 5
idx: 6
idx: 7
idx: 8
idx: 9
```

```
idx: 10
idx: 11
idx: 12
idx: 13
idx: 14
idx: 15
idx: 16
idx: 17
idx: 18
```

In [67]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

def tokenize(text):
    return word_tokenize(text)

max_features = 200
tfidfVectorizer = TfidfVectorizer(input="content", use_idf=True,
                                  tokenizer=tokenize, max_features=max_features,
                                  stop_words="english")


tfidf = tfidfVectorizer.fit_transform(processed_corpus.values())

tfidf_tokens = tfidfVectorizer.get_feature_names_out()

final_vectors = pd.DataFrame(
    data = tfidf.toarray(),
    columns = tfidf_tokens
)
final_vectors
```

```
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/sklearn/feature_extraction/text.py:525: UserW
arning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

Out[67]:

	1	111	2	3	4	able	activity	adult	advice	affect	...	white	woman	work	worse	year	yo
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.03637	0.00000	0.020419	0.022037	...	0.0	0.000000	0.000000	0.000000	0.000000	0.032
1	0.011828	0.000000	0.000000	0.013468	0.016859	0.000000	0.00000	0.00000	0.036549	0.000000	...	0.0	0.000000	0.012683	0.000000	0.000000	0.014
2	0.125821	0.000000	0.063157	0.143263	0.000000	0.000000	0.00000	0.00000	0.048598	0.000000	...	0.0	0.000000	0.000000	0.000000	0.000000	0.077
3	0.000000	0.248377	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.000000	...	0.0	0.000000	0.000000	0.000000	0.000000	0.000
4	0.000000	0.000000	0.016905	0.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.000000	...	0.0	0.024549	0.018056	0.021522	0.000000	0.020
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
973	0.033728	0.134896	0.000000	0.038403	0.048071	0.036776	0.00000	0.00000	0.026054	0.112475	...	0.0	0.000000	0.000000	0.000000	0.000000	0.041
974	0.000000	0.192250	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.037132	0.080148	...	0.0	0.000000	0.000000	0.000000	0.000000	0.000
975	0.000000	0.000000	0.061306	0.069532	0.087038	0.066586	0.00000	0.00000	0.094349	0.025456	...	0.0	0.000000	0.098217	0.039025	0.035993	0.000
976	0.031548	0.000000	0.063343	0.035921	0.000000	0.000000	0.00000	0.04648	0.024371	0.000000	...	0.0	0.000000	0.033827	0.000000	0.000000	0.000
977	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.056559	...	0.0	0.000000	0.024247	0.000000	0.000000	0.027

978 rows × 200 columns

4. Perform integer encoding and one-hot encoding on one of the pre-processed data files and save the output to a txt file. (30 points)

In [80]:
```
import numpy as np
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
label_encoder = LabelEncoder()
keys = list(processed_corpus.keys())
tokens = tokenize(processed_corpus[keys[0]])
integer_encoded = label_encoder.fit_transform(tokens)
print(integer_encoded)
```

```
[ 13  60  97  79  37  26  44  61  99  69  41  40  90 111  46  65  50  44
  53   5 110  10   9  44  79  69   9  44  79  69   4  61  18  74  26  44
   6  89  83 120  93 105 116  24 122   7 104  62   9  15   3   1  61  44
  43  79  69  40  25 115 102 127 113 126   8  35  96   0  22 103  44  61
  74 109  56 114  34  71  49  61  18  74   6  89  69   9  15   3 125  14
  40  29  61  44  14  40  29  61  44  12  40  68  28  51  69  55   2 119
  59 112  29   1  61  44  66  72  30 117  54  36  74  63  61  44  98 108
 100  61  44  42  53  86  78  73  32  79  61  44  69  45  37 104  87  47
  76  87  29   6  89  67  54   6  89  79  70 121  40  53  21  31  65  52
  57  31  38  94  88  97  79  61  44  31 123  29  44  65   5  38  42  53
  64  20  10  11  29  61  44  11  29  61  44  16  29  61  44  79  69  39
  61  77  81  23 101  48 118  26  61  44  74  80  75   6  89  85   1  61
  44  84  92  74  77 124 106  44  19  58  17  28  41  33  61  56  91 128
  87  27  82  95 107 101  48  26  44]
```

In [86]:
```
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(
    len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(
    integer_encoded)
print(onehot_encoded)
pd.DataFrame(onehot_encoded).to_csv("week7_4.txt", sep='\t', index=False)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
/Users/shireesh/opt/anaconda3/envs/COMP293/lib/python3.10/site-packages/sklearn/preprocessing/_encoders.py:975: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

In [77]:
```
inverted = label_encoder.inverse_transform(
    [argmax(onehot_encoded[0, :])])
print(inverted)
```

```
['Research']
```

5. Choose an appropriate word and find the words that are the most similar to it in one of the pre-processed data files. (30 points)

In [98]:
```
import gensim
import nltk
from nltk.corpus import movie_reviews
from gensim.models import Word2Vec
# make a list of movie review documents
documents = [text.split() for text in processed_corpus.values()]
model = Word2Vec(documents, min_count=5)
model.wv.most_similar(positive = ['cancer'],topn = 25)
```

Out[98]:
```
[('breast', 0.9294400811195374),
 ('penile', 0.9016801714897156),
 ('nonmelanoma', 0.9012202024459839),
 ('testicular', 0.8954606056213379),
 ('nasopharyngeal', 0.8859248757362366),
 ('laryngeal', 0.8493385314941406),
 ('prostate', 0.8281572461128235),
 ('Prostate', 0.8127396106719971),
 ('ovarian', 0.7976941466331482),
 ('sinus', 0.7793684601783752),
 ('Types', 0.777524471282959),
 ('melanoma', 0.7684705853462219),
 ('vulval', 0.7663765549659729),
 ('polyp', 0.7647110223770142),
 ('bowel', 0.7627311944961548),
 ('tumour', 0.7616214752197266),
 ('Hodgkin', 0.7559468150138855),
 ('lymphoma', 0.7526421546936035),
 ('cancerous', 0.7447134256362915),
 ('nonHodgkin', 0.7429410815238953),
 ('advanced', 0.7410709857940674),
 ('squamous', 0.7305600643157959),
 ('myeloma', 0.7272151112556458),
 ('radiotherapy', 0.7244316339492798),
 ('pilonidal', 0.719566822052002)]
```