```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
# Imports
import nltk
import ast
import numpy as np
nltk.download('wordnet')
nltk.download('sentiwordnet')
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn
```

```
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data] Downloading package sentiwordnet to /root/nltk_data...
    [nltk_data]   Unzipping corpora/sentiwordnet.zip.
```

```
# synset -> id
synset = wn.synset('organ.N.02')
synsetid = synset.offset()
synsetid
```

```
    8349350
```

```
# wn.synsets('car')[0].hyponyms()
wn.synsets('be')[-1].lemma_names()
```

```
    ['cost', 'be']
```

```
print(wn.synset('organ.N.05').definition())
```

```
    wind instrument whose sound is produced by means of pipes arranged in sets supplied with air from a bellows and controlled from a lar
```

```
import pandas as pd
```

```
df=pd.read_csv('/content/drive/MyDrive/merged_word_net_v3.1.csv')
```

```
df.shape
```

```
    (103860, 10)
```

```
df.head()
```

	Unnamed: 0	POS	swnID	PosScore	NegScore	ObjScore	swnTerm	gloss	WSD	
0	0	a	1740	0.125	0.00	0.875	able#1	(usually followed by `to') having the necessar...	['usually', 'follow', 'have', 'necessary', 'me...	['
1	1	a	2098	0.000	0.75	0.250	unable#1	(usually followed by `to') not having the nece...	['usually', 'follow', 'not', 'have', 'necessar...	[
2	2	a	2312	0.000	0.00	1.000	dorsal#2	facing away from the axis of	['face', 'away', 'axis'	

```
df.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 103860 entries, 0 to 103859
    Data columns (total 10 columns):
     #   Column      Non-Null Count   Dtype  
    ---  ------      --------------   -----  
     0   Unnamed: 0  103860 non-null  int64  
     1   POS         103860 non-null  object 
     2   swnID       103860 non-null  int64  
     3   PosScore    103860 non-null  float64
     4   NegScore    103860 non-null  float64
     5   ObjScore    103860 non-null  float64
     6   swnTerm     103860 non-null  object 
     7   gloss       103860 non-null  object 
     8   WSD         101091 non-null  object 
     9   swnGlossID  101091 non-null  object 
    dtypes: float64(3), int64(2), object(5)
    memory usage: 7.9+ MB
```

```
#Splitting the swnTerm into words
for j in range(len(df['swnTerm'])):
  sentence = df['swnTerm'][j]
  words = sentence.split()
  df['swnTerm'][j]=words
```

```
<ipython-input-11-67d869cfcd9d>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a
  df['swnTerm'][j]=words
```

```
#Reading all distinct synsets present in the glosses
lst={}
for j in range(len(df)):
 if type(df['swnGlossID'][j]) is float:
  continue
 actual_list = ast.literal_eval(df['swnGlossID'][j])
 for word in actual_list:
  lst[word]=1
```

```
len(lst)
```

```
65588
```

```
#Generating synset_names using swn Terms
lst2={}
for j in range(len(df['swnTerm'])):
  if type(df['swnGlossID'][j]) is float:
    continue
  pos = df['POS'][j]
  if pos != 'a':
    pos=chr(ord(pos)-ord('a')+ord('A'))
  actual_list = ast.literal_eval(df['swnGlossID'][j])
  for word in df['swnTerm'][j]:
    sense_num=word[-1]
    word=word[0:-2]
    word=word+"."+pos+".0"+sense_num
    lst2[word]=actual_list
```

```
len(lst2)
```

```
177531
```

```
#Identify synsets whose gloss are not present
gloss_missing=[]
for key in lst.keys():
  if lst2.get(key)==None:
    gloss_missing.append(key)
```

```
#We will ignore these when making 2nd order concept vectors
len(gloss_missing)
```

```
15003
```

```
#Marking these to be absent in terms of gloss defintion
for key in gloss_missing:
  lst2[key]="GLOSS_ABSENT"
```

```
for key in lst.keys():
  lst[key]=lst2[key]
```

```
len(lst)
```

```
65588
```

```
#list of all stopwords
with open('/content/drive/MyDrive/stopwords.txt') as f:
    stopwords = [line.rstrip('\n') for line in f]
```

```
len(lst)
```

```
65588
```

```
#Assigning dimension indices to synsets
dim={}
j=0
for key in lst.keys():
  dim[key]=j
  j=j+1
```

```
#Get sentiment scores form synset name
def get_sentiment_scores_from_synset_name(syn_name):
  try:
    synset = swn.senti_synset(syn_name)
    if synset:
          positive_score = synset.pos_score()
          negative_score = synset.neg_score()
          objective_score = synset.obj_score()
          return positive_score, negative_score, objective_score
  except:
    return None
```

```
#pos capitalizer in synset name
def pos_cap(synset_name):
  i=synset_name.find('.')
  pos=synset_name[i+1]
  if pos!='a':
   pos=chr(ord(pos)-ord('a')+ord('A'))
  synset_name=synset_name[:i+1]+pos+synset_name[i+2:]
  return synset_name
```

```
#Get synset name of the first sense of a word
def first_synset_name_approximation(word):
    synsets = wn.synsets(word)
    if synsets:
        first_synset_id = synsets[0].name()  # Get the name of the first synset
        i=first_synset_id.find('.')
        pos=first_synset_id[i+1]
        pos=chr(ord(pos)-ord('a')+ord('A'))
        first_synset_id=first_synset_id[:i+1]+pos+first_synset_id[i+2:]
        return first_synset_id
    else:
        return None
```

```
word = "hate"
scores = get_sentiment_scores_from_synset_name(first_synset_name_approximation(word))
if scores:
    positive, negative, objective = scores
    print("Positive Score:", positive)
    print("Negative Score:", negative)
    print("Objective Score:", objective)
else:
    print("No sentiment scores found for the word.")
```

```
Positive Score: 0.125
Negative Score: 0.375
Objective Score: 0.5
```

```
#Scoring function defined on the basis of positive negative and obj scores
def sentiment_scoring_function(synset_name):
    scores = get_sentiment_scores_from_synset_name(synset_name)
    if scores:
        positive, negative, objective = scores
        diff = positive - negative
        sign = 1 if diff >= 0 else -1
        return sign * (1 + abs(diff))
    else:
        return 0
```

```
def first_order_concept_vector(synset_name,v):
    if lst.get(synset_name) != None:
        gloss_list=[]
        if lst[synset_name] == 'GLOSS_ABSENT':
            gloss_list.append(synset_name)
        else:
            for c in lst[synset_name]:
                gloss_list.append(c)
            gloss_list.append(synset_name)
    else:
        return v
    # print("First Order:",gloss_list)
    for syns in gloss_list:
        if dim.get(syns)!=None:
            p=sentiment_scoring_function(syns)
            k=v[dim[syns]]+p
            v[dim[syns]]=k
    return v
```

```
def second_order_concept_vector(synset_name,v):
    all=[synset_name]
    # if lst.get(synset_name)!=None:
    if lst.get(synset_name)!=None:
        gloss_list=[]
        # if lst[synset_name]=='GLOSS_ABSENT':
        if lst[synset_name]=='GLOSS_ABSENT':
            gloss_list.append(synset_name)
        else:
            # for c in lst[synset_name]:
            for c in lst[synset_name]:
                gloss_list.append(c)
            gloss_list.append(synset_name)
    else:
        return v
    if gloss_list != "GLOSS_ABSENT":
        for syns in gloss_list:
            all.append(syns)
    # print()
    # print("Second Order:",all)
    for syns in all:
        v=first_order_concept_vector(syns,v)
    return v
```

```
def context_vector(synset_name):
    v=np.zeros(len(lst),dtype=float)
    synset=wn.synset(synset_name)
    i=synset_name.find('.')
    pos=synset_name[i+1]
    all=[]
    all.append(synset_name)
    if pos=='N':
        l1=synset.hypernyms()
        l2=synset.hyponyms()
        if len(l2) > 10:
            l2 = []
        l3=synset.part_meronyms()
        l4=synset.substance_meronyms()
        l5=synset.member_meronyms()
        for syn in l1:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l2:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l3:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l4:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l5:
            all.append(pos_cap(str(syn)[8:-2]))
    elif pos=='V':
        l1=synset.hypernyms()
        l2=synset.hyponyms()
        if len(l2) > 10:
            l2 = []
        l3=synset.entailments()
        l5=synset.causes()
        for syn in l1:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l2:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l3:
            all.append(pos_cap(str(syn)[8:-2]))
        for syn in l5:
            all.append(pos_cap(str(syn)[8:-2]))
    elif pos=='R':
        l1=synset.similar_tos()
        for syn in l1:
            all.append(pos_cap(str(syn)[8:-2]))
    elif pos=='a':
        l1=synset.similar_tos()
        for syn in l1:
            all.append(pos_cap(str(syn)[8:-2]))
    # if len(all) > 10:
    #     all = [synset_name]
    # print("All Hierarchies combines: ",all)
    for syns in all:
        v = second_order_concept_vector(syns,v)
    return v
```

```
v = context_vector("apple.N.01")
# lst["red.a.01"]
```

```
All Hierarchies combines:  ['apple.N.01', 'edible_fruit.N.01', 'pome.N.01', 'cooking_apple.N.01', 'crab_apple.N.03', 'eating_apple.N.

Second Order: ['apple.N.01', 'fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.
First Order: ['fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.01', 'whitish.a
First Order: ['ripened.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'fruit.N.01']
```

```
    First Order: ['red.a.01']
    First Order: ['yellow.a.01']
    First Order: ['green.a.01']
    First Order: ['outer.a.01', 'surface.N.01', 'usually.R.01', 'thin.a.01', 'skin.N.02']
    First Order: ['sweet.a.01']
    First Order: ['taste.V.01', 'sour.N.00', 'lemon.N.04', 'tart.a.01']
    First Order: ['something.N.01', 'see.V.01', 'hear.V.01', 'clearly.R.01', 'define.V.04', 'crisp.a.01']
    First Order: ['whitish.a.01']
    First Order: ['soft.a.01', 'moist.a.01', 'part.N.12', 'fruit.N.01', 'flesh.N.03']
    First Order: ['fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.01', 'whitish.a

    Second Order: ['edible_fruit.N.01', 'edible.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'especially.R.01', 'have.V.01
    First Order: ['edible.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'especially.R.01', 'have.V.01', 'sweet.a.01', 'fles
    First Order: ['suitable.a.01', 'use.N.01', 'food.N.01', 'edible.a.01']
    First Order: ['reproductive.a.01']
    First Order: ['body.N.01']
    First Order: ['plant.N.02', 'reproduce.V.03', 'means.N.01', 'seed.N.02', 'not.R.01', 'spore.N.01', 'seed_plant.N.01']
    First Order: ['distinctly.R.03', 'greater.a.01', 'extent.N.01', 'degree.N.02', 'be.V.01', 'common.a.01', 'especially.R.01']
    First Order: ['have.V.01', 'possess.V.02', 'concrete.a.01', 'abstract.a.01', 'sense.N.01', 'have.V.01']
    First Order: ['sweet.a.01']
    First Order: ['soft.a.01', 'tissue.N.01', 'body.N.01', 'vertebrate.a.01', 'mainly.R.01', 'muscle.N.01', 'tissue.N.01', 'fat.a.06', 'f
    First Order: ['edible.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'especially.R.01', 'have.V.01', 'sweet.a.01', 'fles

    Second Order: ['pome.N.01', 'fleshy.a.02', 'fruit.N.01', 'apple.N.01', 'pear.N.01', 'related.a.01', 'fruit.N.01', 'have.V.01', 'seed.N
    First Order: ['fleshy.a.02', 'fruit.N.01', 'apple.N.01', 'pear.N.01', 'related.a.01', 'fruit.N.01', 'have.V.01', 'seed.N.01', 'chambe
    First Order: ['relate.V.02', 'resemble.V.01', 'flesh.N.01', 'fleshy.a.02']
    First Order: ['ripened.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'fruit.N.01']
    First Order: ['fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.01', 'whitish.a
    First Order: ['sweet.a.01', 'juicy.a.01', 'gritty.a.01', 'textured.a.01', 'fruit.N.01', 'available.a.01', 'many.a.01', 'variety.N.01'
    First Order: ['related.a.01']
    First Order: ['ripened.a.01', 'reproductive.a.01', 'body.N.01', 'seed_plant.N.01', 'fruit.N.01']
    First Order: ['have.V.01', 'possess.V.02', 'concrete.a.01', 'abstract.a.01', 'sense.N.01', 'have.V.01']
    First Order: ['small.a.01', 'hard.a.01', 'fruit.N.01', 'seed.N.01']
    First Order: ['english.a.01', 'architect.N.01', 'chambers.N.01']
    First Order: ['be.V.01', 'outside.N.02', 'further.a.01', 'center.N.04', 'outer.a.01']
    First Order: ['relate.V.02', 'resemble.V.01', 'flesh.N.01', 'fleshy.a.02']
    First Order: ['part.N.01']
    First Order: ['fleshy.a.02', 'fruit.N.01', 'apple.N.01', 'pear.N.01', 'related.a.01', 'fruit.N.01', 'have.V.01', 'seed.N.01', 'chambe

    Second Order: ['cooking_apple.N.01', 'apple.N.01', 'use.V.01', 'primarily.R.01', 'cook.V.01', 'pie.N.01', 'applesauce.N.01', 'cooking
    First Order: ['apple.N.01', 'use.V.01', 'primarily.R.01', 'cook.V.01', 'pie.N.01', 'applesauce.N.01', 'cooking_apple.N.01']
    First Order: ['fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.01', 'whitish.a
    First Order: ['use.V.01']
    First Order: ['for_the_most_part.R.01', 'primarily.R.01']
    First Order: ['prepare.V.01', 'hot.a.01', 'meal.N.01', 'cook.V.01']
    First Order: ['dish.N.02', 'bake.V.01', 'pastry.N.02', 'lined.a.01', 'pan.N.01', 'often.R.01', 'pastry.N.02', 'top.a.01', 'pie.N.01']
    First Order: ['puree.N.01', 'stewed.a.01', 'apple.N.01', 'usually.R.01', 'sweetened.a.01', 'spice.V.00', 'applesauce.N.01']
    First Order: ['apple.N.01', 'use.V.01', 'primarily.R.01', 'cook.V.01', 'pie.N.01', 'applesauce.N.01', 'cooking_apple.N.01']

    Second Order: ['eating_apple.N.01', 'apple.N.01', 'use.V.01', 'primarily.R.01', 'eat.V.01', 'raw.N.01', 'cooking.N.01', 'eating_apple
    First Order: ['apple.N.01', 'use.V.01', 'primarily.R.01', 'eat.V.01', 'raw.N.01', 'cooking.N.01', 'eating_apple.N.01']
    First Order: ['fruit.N.01', 'red.a.01', 'yellow.a.01', 'green.a.01', 'skin.N.02', 'sweet.a.01', 'tart.a.01', 'crisp.a.01', 'whitish.a
    First Order: ['use.V.01']
```

```
np.count_nonzero(v)
```

```
    113
```

```
for idx in range(len(v)):
    if v[idx] != 0:
        print(str(idx).ljust(10), v[idx])
```

```
    0          2.0
    2          10.0
    4          1.0
    10         -1.625
    24         1.0
    27         7.0
    29         3.0
    42         6.0
    60         2.25
    91         1.0
    104        3.0
    105        1.0
    111        2.0
    120        3.0
    133        2.0
    150        2.75
    151        2.0
    267        2.0
    269        1.25
    285        -1.25
    393        1.0
    558        -2.5
    568        1.125
    597        1.0
    619        1.0
    671        3.0
    1148       1.0
    1183       -1.375
    1216       1.0
    1472       1.0
    1493       1.0
    1624       1.0
    1850       1.0
    1937       1.375
```

```
1975       2.0
2058      -1.75
2155       7.5
2411       6.0
2635       1.0
2693       1.0
2730       1.0
2994      15.0
3005      -6.75
3113       4.0
3413       6.0
3506       1.0
3830       1.0
4192       1.875
4338       1.0
4400      -1.25
4556       2.0
5000       3.0
5012       3.75
5558       3.0
5852       1.0
5998       1.0
6121       5.0
6307      -1.125
```

Generation

```
def find_vector(lyrics):
    lyrics = lyrics.replace(',', ' ').replace('.', ' ').replace('-', '_').lower()
    lyrics_split = lyrics.split()
    lyrics_split = [elem for elem in lyrics_split if elem not in stopwords]
    word_vectors = np.zeros(65588)

    # create word_vectors dictionary
    for word in lyrics_split:
        vector = nltk.wsd.lesk(lyrics_split, word)

        if vector is None:
            continue
        else:
            vector = str(vector)[8:-2]
            vector = vector.replace('.n.', '.N.').replace('.r.', '.R.').replace('.v.', '.V.').replace('.s.', '.a.')

        word_vectors += context_vector(vector)

    # Generate code
    return word_vectors
```

```
from tqdm import tqdm
gpt2 = pd.read_csv('/content/drive/MyDrive/GPT2_summaries.csv')
```

```
gpt2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5925 entries, 0 to 5924
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   index    5925 non-null   int64 
 1   title    5925 non-null   object
 2   summary  5925 non-null   object
dtypes: int64(1), object(2)
memory usage: 139.0+ KB
```

```
count = 0
lyrics_vector = np.zeros(shape=(5925, 65588))

for _, row in tqdm(gpt2.iterrows(), total=5925):
    lyrics_vector[count] = find_vector(row['summary'])
    count += 1
```

Save original array in npy (2.9GB)

```
np.save('/content/drive/MyDrive/lyrics_vectors.npy', lyrics_vector)
```

```
vector_lst = np.load('/content/drive/MyDrive/lyrics_vector.npy')
# index_data = np.load('/content/drive/MyDrive/lyrics_vector_index.npy')
```

```
print(len(vector_lst[0]))
print(len(vector_lst[1]))
```

```
65588
65588
```

Save array in sparse matrix in npz (88MB)

```
from scipy.sparse import csr_matrix
```

```
# Initialize lists to store data, indices, and indptr for each row
all_data = []
all_indices = []
all_indptr = [0]  # Start with zero

# Process each row in the 2D array
for row in lyrics_vector:
    # Indices of non-zero elements in the current row
    nonzero_indices = np.nonzero(row)[0]

    # Values of non-zero elements in the current row
    nonzero_values = row[nonzero_indices]

    # Update indptr for the next row
    all_indptr.append(all_indptr[-1] + len(nonzero_indices))

    # Append data and indices
    all_data.extend(nonzero_values)
    all_indices.extend(nonzero_indices)

# Create CSR matrix
sparse_matrix = csr_matrix((all_data, all_indices, all_indptr), shape=(len(lyrics_vector), 65588))

# Save the CSR matrix to file
np.savez('sparse_vector_lst.npz', data=sparse_matrix.data, indices=sparse_matrix.indices, indptr=sparse_matrix.indptr, shape=sparse_matrix
```

```
# Load CSR matrix from file
loaded_data_2d = np.load('sparse_vector_lst.npz')
sparse_matrix_loaded = csr_matrix((loaded_data_2d['data'], loaded_data_2d['indices'], loaded_data_2d['indptr']), shape=loaded_data_2d['sh

# Convert it back to 2D array
vector_lst = sparse_matrix_loaded.toarray()
```