```
# documents
doc1, doc2='I like dogs.', 'I hate dogs.'

# Split the documents and create tokens
doc1_tokens=set(doc1.lower().split())
doc2_tokens=set(doc2.lower().split())

#Print the tokens
print(doc1_tokens,doc2_tokens)


# Calculate the Jaccard Similarity
print(len(doc1_tokens.intersection(doc2_tokens)))
print(len(doc1_tokens.union(doc2_tokens)))
print(doc1_tokens.union(doc2_tokens))
jaccard_similarity=
len(doc1_tokens.intersection(doc2_tokens))/len(doc1_tokens.union(doc2_
tokens))

# Print the Jaccard Simialrity score
print(jaccard_similarity)

# I like dogs hate
# s1: I like dogs.->1 1 1 0 (one hot encoding)
# s2: I hate dogs->1 0 1 1


from scipy.spatial import distance
distance.dice([1,1,1,0],[1,0,1,1])
```

#Cosine Similarity using Spacy

```
import en_core_web_sm

nlp = en_core_web_sm.load()

## try medium or large spacy english models

doc1 = nlp("I like apples.")

doc2 = nlp("I like oranges.")

# cosine similarity
doc1.similarity(doc2)


doc1 = nlp("I like dogs.")
```

```
doc2 = nlp("I hate dogs.")

# cosine similarity
doc1.similarity(doc2)
```

#Cosine Similarity using Scipy

```
from scipy import spatial

# Document Vectorization
doc1, doc2 = nlp('I like apples.').vector, nlp('I like
oranges.').vector
print(doc1)
print(type(doc1))
print(doc1.shape)

print(doc2)

# Cosine Similarity
result = 1 - spatial.distance.cosine(doc1, doc2)

print(result)
```

#Let’s create a search engine using Text Similarity measures

```
import en_core_web_sm
from numpy import dot
from numpy.linalg import norm
import numpy as np
import pandas as pd

nlp = en_core_web_sm.load()


# Prepare dataset
doc_list=['I love this sandwich.',
          'this is an amazing place!',
          'I feel very good about these beers.',
          'this is my best work.',
          'what an awesome view',
          'I do not like this restaurant',
          'I am tired of this stuff.',
          "I can't deal with this",
          'he is my sworn enemy!',
          'my boss is horrible.',
          'I hate this sandwich.']

# user input
query=input()
```

```
# similarity score
sim_scores=[]

# Vectorize input query
q_vector=nlp(query).vector

for doc in doc_list:
    # Vectorize document
    doc_vector=nlp(doc).vector
    # Cosine Similarity
    cos_sim = dot(q_vector,
doc_vector)/(norm(q_vector)*norm(doc_vector))
    # append the score
    sim_scores.append(cos_sim)

# most similar
most_similar=doc_list[sim_scores.index(max(sim_scores))]
print("\nMost Similar:\n",most_similar)

# sorting most similar sentences
top_index=list(np.argsort(sim_scores)[-5:]) #argsort() by default
sorts in ascending order , therefore  this part selects the last 5
indices from the sorted list. Since the indices are sorted in
ascending order of similarity scores, selecting the last 5 indices
effectively gives us the indices corresponding to the top 5 most
similar sentences.

# [0.8, 0.4, 0.9, 0.45]
# argsort: [0.4,0.45, 0.8, 0.9]-> 1,3,0,2

print(top_index)
top_index.reverse()
print(top_index)

print("\nMost Similar Documents:\n")
for i in top_index:
    print(doc_list[i],sim_scores[i])

print(sim_scores)
```

#Training Word2vec model

```
import re  # For preprocessing
import pandas as pd  # For data handling (alias)
from time import time  # To time our operations
from collections import defaultdict  # For word frequency

import spacy  # For preprocessing

import logging  # Setting up the loggings to monitor gensim
logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s",
                    datefmt= '%H:%M:%S', level=logging.INFO)

df=pd.read_csv('/content/simpsons_script_lines.csv')
df.head()

<ipython-input-2-ee713851d592>:1: DtypeWarning: Columns (4,5,6) have
mixed types. Specify dtype option on import or set low_memory=False.
  df=pd.read_csv('/content/simpsons_script_lines.csv')

{"type":"dataframe","variable_name":"df"}

df.columns

Index(['id', 'episode_id', 'number', 'raw_text', 'timestamp_in_ms',
       'speaking_line', 'character_id', 'location_id',
'raw_character_text',
       'raw_location_text', 'spoken_words', 'normalized_text',
'word_count'],
      dtype='object')

df=df[['raw_character_text','spoken_words']]

df

{"type":"dataframe","variable_name":"df"}

df.isnull().sum()

raw_character_text    17522
spoken_words          26159
dtype: int64

# nlp = spacy.load('en', disable=['ner', 'parser'])
nlp = spacy.load("en_core_web_sm")

def cleaning(doc):
    # Lemmatizes and removes stopwords
    # doc needs to be a spacy Doc object
    txt = [token.lemma_ for token in doc if not token.is_stop]
    if len(txt) > 2:
        return ' '.join(txt)
```

```
brief_cleaning = (re.sub("[^A-Za-z']+", ' ', str(row)).lower() for row
in df['spoken_words'])
```

```
# t = time()
# txt = [cleaning(doc) for doc in nlp.pipe(brief_cleaning,
batch_size=5000,
#                    n_threads=-1)]
# print('Time to clean up everything: {} mins'.format(round((time() -
t) / 60, 2)))
t = time()
txt = [cleaning(doc) for doc in nlp.pipe(brief_cleaning,
batch_size=5000)]
print('Time to clean up everything: {} mins'.format(round((time() - t)
/ 60, 2)))
```

```
Time to clean up everything: 4.09 mins
```

```
df_clean = pd.DataFrame({'clean': txt})
df_clean = df_clean.dropna().drop_duplicates()
df_clean.shape
```

```
(86211, 1)
```

```
from gensim.models.phrases import Phrases, Phraser
sent = [row.split() for row in df_clean['clean']]
```

```
phrases = Phrases(sent, min_count=30, progress_per=10000)
```

```
phrases
```

```
<gensim.models.phrases.Phrases at 0x79b0b11c0430>
```

```
bigram = Phraser(phrases)
```

```
# I love ice cream
```

```
sentences = bigram[sent]
```

```
word_freq = defaultdict(int)
for sent in sentences:
    for i in sent:
        word_freq[i] += 1
len(word_freq)
```

```
29791
```

```
sorted(word_freq, key=word_freq.get, reverse=True)[:10]
```

```
['oh', 'like', 'know', 'get', 'hey', 'think', 'come', 'right', 'look',
'want']
```

```
import multiprocessing

from gensim.models import Word2Vec

cores = multiprocessing.cpu_count() # Count the number of cores in a
computer


w2v_model = Word2Vec(min_count=20,
                     window=2,
                     vector_size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)

t = time()

w2v_model.

print('Time to build vocab: {} mins'.format(round((time() - t) / 60,
2)))
```

```
Time to build vocab: 0.01 mins
```

```
I like dogs and cats and rabbits
1 2     3   4   5    4   6

{I: 1, like: 2..}

t = time()
# print(total_examples)

w2v_model.train(sentences, total_examples=w2v_model.corpus_count,
epochs=30, report_delay=1)

print('Time to train the model: {} mins'.format(round((time() - t) /
60, 2)))
```

```
Time to train the model: 1.39 mins
```

```
w2v_model.corpus_count
```

```
86211
```

```
sentences
```

```
<gensim.interfaces.TransformedCorpus at 0x79b0a94e8130>
```

```
w2v_model.init_sims(replace=True)
```

```
<ipython-input-47-c7757d71a30b>:1: DeprecationWarning: Call to
deprecated `init_sims` (Gensim 4.0.0 implemented internal
optimizations that make calls to init_sims() unnecessary. init_sims()
is now obsoleted and will be completely removed in future versions.
See https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-
Gensim-3.x-to-4).
  w2v_model.init_sims(replace=True)
WARNING:gensim.models.keyedvectors:destructive init_sims(replace=True)
deprecated & no longer required for space-efficiency
```

```
similar_words = w2v_model.wv.most_similar(positive=["homer"])
for word, similarity in similar_words:
    print(f"{word}: {similarity}")
```

```
gee: 0.8816429376602173
quick: 0.8771034479141235
sweetheart: 0.867999792098999
marge: 0.8667840957641602
hopeless: 0.862589418888092
glad: 0.8622774481773376
bartender: 0.8606762886047363
right: 0.858879029750824
sorry: 0.8574423789978027
help: 0.8572109937667847
```

```
w2v_model.wv.most_similar(positive=["homer_simpson"])
```

```
[('select', 0.9020684361457825),
 ('lady_gentleman', 0.8911157846450806),
 ('trial', 0.887400209903717),
 ('dedicate', 0.8839735388755798),
 ('crew', 0.8823495507240295),
 ('congratulation', 0.8809747695922852),
 ('broadcast', 0.8807786703109741),
 ('convention', 0.879878044128418),
 ('direct', 0.8789938688278198),
 ('host', 0.8774415254592896)]
```

```
w2v_model.wv.most_similar(positive=["marge"])
```

```
[('glad', 0.8914166688919067),
 ('sure', 0.8864542245864868),
 ('pregnant', 0.8832345008850098),
 ('homie', 0.8820032477378845),
 ('talk', 0.8817054033279419),
 ('advice', 0.8815292716026306),
 ('worried', 0.8806873559951782),
 ('someday', 0.8791307806968689),
 ('gee', 0.8790706396102905),
 ('darling', 0.8790403604507446)]
```

```
w2v_model.wv.most_similar(positive=["bart"])
```

```
[('mom_dad', 0.9176613688468933),
 ('lisa', 0.9168039560317993),
 ('upset', 0.9008905291557312),
 ('worried', 0.9008179903030396),
 ('mother', 0.898597776889801),
 ('concerned', 0.8945304751396179),
 ('admit', 0.8920857906341553),
 ('brother', 0.8900191783905029),
 ('worry', 0.8900096416473389),
 ('humiliate', 0.8885989785194397)]
```

```
w2v_model.wv.similarity('maggie', 'baby')
```

```
0.7319323
```

```
w2v_model.wv.similarity('bart', 'nelson')
```

```
0.81326276
```

```
w2v_model.wv.doesnt_match(['jimbo', 'milhouse', 'kearney'])
```

```
WARNING:gensim.models.keyedvectors:vectors for words {'kearney'} are
not present in the model, ignoring these words
```

```
{"type":"string"}
```

```
w2v_model.wv.doesnt_match(['homer', 'patty', 'selma'])
```

```
{"type":"string"}
```

```
w2v_model.wv.most_similar(positive=["woman", "homer"],
negative=["marge"], topn=3)
```

```
[('screw', 0.844830334186554),
 ('effort', 0.8303388357162476),
 ('attractive', 0.8162267208099365)]
```

```
w2v_model.wv.most_similar(positive=["woman", "bart"],
negative=["man"], topn=3)
```

```
[('important', 0.821419358253479),
 ('understand', 0.809717059135437),
 ('know', 0.8067172765731812)]
```

```
w2v_model.wv.most_similar(positive=["woman", "queen"],
negative=["man"], topn=3)
```

```
[('spelling', 0.5295670628547668),
 ('king', 0.5204694271087646),
 ('pageant', 0.5008288025856018)]

# w2v_model.wv.most_similar(a=["woman", "queen"], b=["man"], topn=3)

----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-31-e3aac44ae70c> in <cell line: 1>()
----> 1 w2v_model.wv.most_similar(a=["woman", "queen"], b=["man"],
topn=3)

TypeError: KeyedVectors.most_similar() got an unexpected keyword
argument 'a'
```