Mssv: 19522348

Tên: Lê Đức Tín

Github: https://github.com/tinld/MKTG5883.N22.CTTT

Lab 6

1. Text Normalization

```
# import needed libraries
import nltk
import numpy as np
import pandas as pd
```

+ Mã + Văn bản

```
df=pd.read_csv("/content/drive/MyDrive/Colab/elonmusk_tweets.csv")
```

```
df.shape
```

```
(2819, 3)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2819 entries, 0 to 2818
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          2819 non-null   int64 
 1   created_at  2819 non-null   object
 2   text        2819 non-null   object
dtypes: int64(1), object(2)
memory usage: 66.2+ KB
```

```
df.describe()
```

	id
count	2.819000e+03
mean	5.804848e+17
std	2.186404e+17
min	1.543473e+10
25%	3.506818e+17
50%	6.569719e+17
75%	7.704732e+17
max	8.496369e+17

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
import string
```

```
def normalize(document):
    # Remove punctuation
    text = "".join([ch for ch in document if ch not in string.punctuation])

    # Tokenize text
```

```
        tokens = word_tokenize(text)

        # Stemming
        stemmer = PorterStemmer()
        ret = " ".join([stemmer.stem(word.lower()) for word in tokens])

        return ret

    original_documents = [x.strip() for x in df['text']]
    documents = [normalize(d).split() for d in original_documents]
    documents[0]
```

```
['band', 'robot', 'spare', 'human', 'httpstcov7jujqwfcv']
```

```
    import re

    emoticons_str = r"""
        (?:
            [:=;] # Eyes
            [oO\-]? # Nose (optional)
            [D\)\]\(\]/\\OpP] # Mouth
        )"""

    regex_str = [
        emoticons_str,
        r'<[^>]+>', # HTML tags
        r'(?:@[\w_]+)', # @-mentions
        r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
        r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&amp;+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+', # URLs

        r'(?:(?:\d+,?)+(?:\.?\d+)?)', # numbers
        r"(?:[a-z][a-z'\-_]+[a-z])", # words with - and '
        r'(?:[\w_]+)', # other words
        r'(?:\S)' # anything else
    ]

    tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORECASE)
    emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASE)

    def tokenize(s):
        return tokens_re.findall(s)

    def preprocess(s, lowercase=False):
        tokens = tokenize(s)
        if lowercase:
            tokens = [token if emoticon_re.search(token) else token.lower() for token in tokens]
        return tokens
    original_documents = [x.strip() for x in df['text']]
    documents = [preprocess(d) for d in original_documents]

    documents[1]
```

```
['b',
 '"',
 '@ForIn2020',
 '@waltmossberg',
 '@mims',
 '@defcon_5',
 'Exactly',
 '.',
 'Tesla',
 'is',
 'absurdly',
 'overvalued',
 'if',
 'based',
 'on',
 'the',
 'past',
 ',',
 'but',
 "that's",
 'irr',
 '\\',
 'xe2',
 '\\',
 'x80',
 '\\',
```

```
        'xa6',
        'https://t.co/qQcTqkzgMl',
        '""']
```

2. Implement TF-IDF

```
import pandas as pd
from collections import Counter
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
import math


#Flatten all the documents
flat_list = [word for doc in documents for word in doc]
#TODO: remove stop words from the vocabulary
words = [word for word in flat_list if word not in stopwords.words('english')]
# TODO: we take the 500 most common words only
counts = Counter (words)
vocabulary = counts.most_common (500)
print([x for x in vocabulary if x[0] == 'Tesla'])
vocabulary =  [x[0] for x in vocabulary]
assert len (vocabulary) == 500
#vocabulary.sort()
vocabulary[:5]
```

```
     [('Tesla', 272)]
     ['.', "'", 'b', '\\', '"']
```

```
def idf(vocabulary, documents):
    idf_values = {}
    num_documents = len(documents)
    for term in vocabulary:
        count = sum(term in document for document in documents)
        idf_values[term] = math.log(num_documents / count, 2)
    return idf_values

idf_values = idf(vocabulary, documents)
[idf_values[key] for key in vocabulary[:5]]
```

```
     [0.959130577668125,
      0.7493007890060756,
      1.1177820471225408,
      3.1034157579523383,
      2.384152165519591]
```

3. Compare the results with the reference implementation of scikit-learn library.

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem.snowball import FrenchStemmer

stemmer = FrenchStemmer()
analyzer = CountVectorizer().build_analyzer()

def stemmed_words(doc):
    return (stemmer.stem(w) for w in analyzer(doc))

# Convert the list of documents into a single string
corpus = [' '.join(doc) for doc in documents]

stem_vectorizer = CountVectorizer(analyzer=stemmed_words)
print(stem_vectorizer.fit_transform(corpus))
print(stem_vectorizer.get_feature_names_out())
```

```
       (0, 617)      1
       (0, 6120)     1
       (0, 6614)     1
       (0, 5674)     1
       (0, 6187)     1
       (0, 3209)     1
       (0, 3203)     1
       (0, 1462)     1
       (0, 7107)     1
```

```
  (1, 6614)     1
  (1, 3203)     1
  (1, 1462)     1
  (1, 2667)     1
  (1, 7259)     1
  (1, 4285)     1
  (1, 1881)     1
  (1, 2392)     1
  (1, 6575)     1
  (1, 3489)     1
  (1, 403)      1
  (1, 4846)     1
  (1, 3278)     1
  (1, 897)      1
  (1, 4751)     1
  (1, 4936)     1
  :     :
  (2817, 2740)  1
  (2817, 3257)  1
  (2817, 5817)  1
  (2817, 3258)  1
  (2817, 1162)  1
  (2817, 7201)  1
  (2817, 5655)  1
  (2817, 7123)  1
  (2817, 4655)  1
  (2817, 5409)  1
  (2818, 3489)  1
  (2818, 6612)  1
  (2818, 6740)  1
  (2818, 6658)  1
  (2818, 441)   1
  (2818, 7272)  1
  (2818, 733)   1
  (2818, 4192)  2
  (2818, 924)   1
  (2818, 5190)  1
  (2818, 6907)  1
  (2818, 5056)  1
  (2818, 6144)  1
  (2818, 3282)  1
  (2818, 5168)  1
['00' '000' '01' ... 'zyfazr2bb2' 'zyv4h85o' 'zzijxxyy']
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
tfidf = TfidfVectorizer (analyzer='word', ngram_range=(1,1), min_df = 1, stop_words = 'english', max_features=500)

features=tfidf.fit(original_documents)
corpus_tf_idf = tfidf.transform (original_documents)

sum_words = corpus_tf_idf.sum(axis=0)
words_freq = [(word, sum_words [0, idx]) for word, idx in tfidf.vocabulary_.items()]
print (sorted (words_freq, key = lambda x: x[1], reverse=True)[:5])
print('testla', corpus_tf_idf [1, features.vocabulary_['tesla']])
```

```
[('http', 163.54366542841234), ('https', 151.85039944652075), ('rt', 112.61998731390989), ('tesla', 95.96401470715628), ('xe2', 88.20944
testla 0.3495243100660956
```

4. Apply TF-IDF for information retrieval

```
from nltk.stem import PorterStemmer
def cosine_similarity (v1, v2):

  sumxx, sumxy, sumyy = 0, 0, 0
  for i in range(len(v1)):
    x = v[i];
    y = v2[i]
    sumxx += x*x
    sumyy += y*y
    sumxy += x*y
  if sumxy == 0:
    result = 0
  else:
    result = sumxy/math.sqrt(sumxx*sumyy)
  return result
```

```
def search_vec (query, k, vocabulary, stemmer, document_vectors, original_documents):
  q= query.split()
  q = [stemmer.stem (w) for w in q]
  query_vector = vectorize(q, vocabulary, idf)
    # TODO: rank the documents by cosine similarity
  scores = [[cosine_similarity (query_vector, document_vectors[d]), d] for d in range(len(document_vectors))]
  scores.sort(key=lambda x: -x[0])

  print('Top-{0} documents'.format(k))
  for i in range(k):
     print(i, original_documents [scores [i][1]])

query= "tesla nasa"
stemmer = PorterStemmer()
document_vectors=words_freq
search_vec(query, 5, vocabulary, stemmer, document_vectors, original_documents)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-9d6b9465bc56> in <cell line: 31>()
     29 query= "tesla nasa"
     30 stemmer = PorterStemmer()
---> 31 document_vectors=words_freq
     32 search_vec(query, 5, vocabulary, stemmer, document_vectors, original_documents)

NameError: name 'words_freq' is not defined
```

SEARCH STACK OVERFLOW

II. Text Processing

1. Preprocessing

```
import nltk
nltk.download('punkt') #Run this line one time to get the resource
nltk.download('stopwords') #Run this line one time to get the resource
nltk.download('wordnet') #Run this line one time to get the resource
nltk.download('averaged_perceptron_tagger') #Run this line one time to get the resource
import numpy as np
import pandas as pd
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
df=pd.read_csv("/content/drive/MyDrive/Colab/coldplay.csv")
```

```
df
```

	Artist	Song	Link	Lyrics
0	Coldplay	Another's Arms	/c/coldplay/anothers+arms_21079526.html	Late night watching tv \nUsed to be you here ...

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Artist  120 non-null    object
 1   Song    120 non-null    object
 2   Link    120 non-null    object
 3   Lyrics  120 non-null    object
dtypes: object(4)
memory usage: 3.9+ KB
None
```

	Weekend			\nYou know you m...

```
song_title = 'Every Teardrop Is A Waterfall'
lyrics = df.loc[df['Song'] == song_title, 'Lyrics'].values[0]
print(lyrics)
```

```
I turn the music up, I got my records on
I shut the world outside until the lights come on
Maybe the streets alight, maybe the trees are gone
I feel my heart start beating to my favourite song

And all the kids they dance, all the kids all night
Until Monday morning feels another life
I turn the music up
I'm on a roll this time
And heaven is in sight

I turn the music up, I got my records on
From underneath the rubble sing a rebel song
Don't want to see another generation drop
I'd rather be a comma than a full stop

Maybe I'm in the black, maybe I'm on my knees
Maybe I'm in the gap between the two trapezes
But my heart is beating and my pulses start
Cathedrals in my heart

As we saw oh this light I swear you, emerge blinking into
To tell me it's alright
As we soar walls, every siren is a symphony
And every tear's a waterfall
Is a waterfall
Oh
Is a waterfall
Oh oh oh
Is a is a waterfall
Every tear
Is a waterfall
Oh oh oh

So you can hurt, hurt me bad
But still I'll raise the flag

Oh
It was a wa wa wa wa wa-aterfall
A wa wa wa wa wa-aterfall

Every tear
Every tear
Every teardrop is a waterfall

Every tear
Every tear
Every teardrop is a waterfall

Every tear
Every tear
Every teardrop is a waterfall
```

```
import nltk
import string
```

```
# Tokenize the lyrics
from nltk import word_tokenize
words = word_tokenize(lyrics)
print(words)
```

```
['I', 'turn', 'the', 'music', 'up', ',', 'I', 'got', 'my', 'records', 'on', 'I', 'shut', 'the', 'world', 'outside', 'until', 'the', 'lig
```

```
tokens_without_punctuation = [token for token in words if token not in string.punctuation]

print(tokens_without_punctuation)
```

```
['I', 'turn', 'the', 'music', 'up', 'I', 'got', 'my', 'records', 'on', 'I', 'shut', 'the', 'world', 'outside', 'until', 'the', 'lights',
```

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

word_tokens = word_tokenize(lyrics)
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in words if not w.lower() in stop_words]
#with no lower case conversion
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(filtered_sentence)
```

```
['I', 'turn', 'music', ',', 'I', 'got', 'records', 'I', 'shut', 'world', 'outside', 'lights', 'come', 'Maybe', 'streets', 'alight', ',',
```

```
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_sentence]
print(lemmatized_tokens)
```

```
['I', 'turn', 'music', ',', 'I', 'got', 'record', 'I', 'shut', 'world', 'outside', 'light', 'come', 'Maybe', 'street', 'alight', ',', 'm
```

```
from nltk import pos_tag
pos_tags = nltk.pos_tag(lemmatized_tokens)

print(pos_tags)
```

```
[('I', 'PRP'), ('turn', 'VBP'), ('music', 'NN'), (',', ','), ('I', 'PRP'), ('got', 'VBD'), ('record', 'NN'), ('I', 'PRP'), ('shut', 'VBF
```

```
from nltk.corpus import wordnet

def get_wordnet_pos(pos_tag):
    output = np.asarray(pos_tag)
    for i in range(len(pos_tag)):
        if pos_tag[i][1].startswith('J'):
            output[i][1] = wordnet.ADJ
        elif pos_tag[i][1].startswith('V'):
            output[i][1] = wordnet.VERB
        elif pos_tag[i][1].startswith('R'):
            output[i][1] = wordnet.ADV
        else:
            output[i][1] = wordnet.NOUN
    return output
```

2. Bag-of-words

```
import nltk
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer


df=pd.read_csv("/content/drive/MyDrive/Colab/coldplay.csv")


print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Artist  120 non-null    object
 1   Song    120 non-null    object
 2   Link    120 non-null    object
 3   Lyrics  120 non-null    object
dtypes: object(4)
memory usage: 3.9+ KB
None
```

```
from sklearn.feature_extraction.text import CountVectorizer



vectorizer = CountVectorizer()

bow = vectorizer.fit_transform(df['Lyrics'])

print( bow.shape)
```

```
(120, 1776)
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer



# Get the feature names (words) from the vectorizer
feature_names = vectorizer.get_feature_names_out()

# Create a dataframe from the BOW matrix and feature names
bow_df = pd.DataFrame(bow.toarray(), columns=feature_names)

bow_df
```

	10	2000	2gether	76543	aaaaaah	aaaaah	aaaah	about	above	achin	...	yellow	yes	yesterday
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
115	0	0	0	0	0	0	0	1	2	0	...	0	0	0
116	0	0	0	0	0	0	0	0	0	0	...	0	0	0
117	0	0	1	0	0	0	0	0	0	0	...	0	0	0
118	0	0	0	0	0	0	0	0	0	0	...	0	0	0
119	0	0	0	0	0	0	0	0	0	0	...	0	0	0

120 rows × 1776 columns

```
sum_bow = bow_df.sum()
sum_bow.idxmax()
```

```
'you'
```

```
word_counts = bow_df.sum()
top_10_words = word_counts.nlargest(10)

# Print the top 10 words
print(top_10_words)
```

```
you    994
the    777
and    650
to     481
it     458
oh     334
in     318
me     314
my     288
on     285
dtype: int64
```

III. Text Similarity

1.. Similarity metrics

```
import nltk
import numpy as np
import pandas as pd
```

```
A = "Outside the classroom, Stallman pursued his studies with even more diligence, rushing off to fulfill his laboratory-assistant duties at
B = "To facilitate the process, AI Lab hackers had built a system that displayed both the source and display modes on a split screen. Despite
C = "With no dorm and no dancing, Stallman's social universe imploded. Like an astronaut experiencing the aftereffects of zero-gravity, Stall
```

```
set_A = set(A.lower().split())
set_B = set(B.lower().split())
set_C = set(C.lower().split())

# Compute the intersection and union
intersection_AB = len(set_A.intersection(set_B))
union_AB = len(set_A.union(set_B))

intersection_AC = len(set_A.intersection(set_C))
union_AC = len(set_A.union(set_C))

intersection_BC = len(set_B.intersection(set_C))
union_BC = len(set_B.union(set_C))

# Compute and print the Jaccard Similarity
jaccard_AB = intersection_AB / union_AB
jaccard_AC = intersection_AC / union_AC
jaccard_BC = intersection_BC / union_BC

print("Jaccard Similarity between A and B:", jaccard_AB)
print("Jaccard Similarity between A and C:", jaccard_AC)
print("Jaccard Similarity between B and C:", jaccard_BC)
```

```
Jaccard Similarity between A and B: 0.08641975308641975
Jaccard Similarity between A and C: 0.12631578947368421
Jaccard Similarity between B and C: 0.0945945945945946
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Compute TF-IDF for sentences A, B, and C
tfidf = vectorizer.fit_transform([A, B, C])

# Calculate cosine similarities
cosine_sim_AB = cosine_similarity(tfidf[0], tfidf[1])
cosine_sim_BC = cosine_similarity(tfidf[1], tfidf[2])
cosine_sim_AC = cosine_similarity(tfidf[0], tfidf[2])

# Print cosine similarities
print("cos(A, B):", cosine_sim_AB)
print("cos(B, C):", cosine_sim_BC)
```

```
print("cos(A, C):", cosine_sim_AC)
```

```
cos(A, B): [[0.1679327]]
cos(B, C): [[0.13618963]]
cos(A, C): [[0.2850296]]
```

2. TF-IDF

```
import nltk
import numpy as np
import pandas as pd
```

```
df=pd.read_csv("/content/drive/MyDrive/Colab/headlines.csv")
```

```
df.head()
```

	publish_date	headline_text
0	20170721	algorithms can make decisions on behalf of fed...
1	20170721	andrew forrests fmg to appeal pilbara native t...
2	20170721	a rural mural in thallan
3	20170721	australia church risks becoming haven for abusers
4	20170721	australian company usgfx embroiled in shanghai...

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1999 entries, 0 to 1998
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   publish_date   1999 non-null   int64 
 1   headline_text  1999 non-null   object
dtypes: int64(1), object(1)
memory usage: 31.4+ KB
None
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import string
df['tokens'] = df['headline_text'].apply(lambda x: word_tokenize(x))

# Remove punctuation
df['tokens'] = df['tokens'].apply(lambda tokens: [token for token in tokens if token not in string.punctuation])

# Remove stop words
stop_words = set(stopwords.words('english'))
df['tokens'] = df['tokens'].apply(lambda tokens: [token for token in tokens if token.lower() not in stop_words])

# Stemming
stemmer = PorterStemmer()
df['tokens'] = df['tokens'].apply(lambda tokens: [stemmer.stem(token) for token in tokens])

# Join tokens back into a single string
df['Stem'] = df['tokens'].apply(lambda tokens: ' '.join(tokens))
```

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

```
df['tokens']
```

```
0        [algorithm, make, decis, behalf, feder, minist]
1      [andrew, forrest, fmg, appeal, pilbara, nativ,...
2                                [rural, mural, thallan]
3                [australia, church, risk, becom, abus]
4      [australian, compani, usgfx, embroil, shanghai...
```

```
                                  ...
1994    [constitut, avenu, win, top, prize, act, archi...
1995                        [dark, mofo, number, crunch]
1996    [david, petraeu, say, australia, must, firm, s...
1997    [driverless, car, australia, face, challeng, r...
1998               [drug, compani, criticis, price, hike]
Name: tokens, Length: 1999, dtype: object
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer=lambda x: x)

vectorizer.fit(df['tokens'])
bow = vectorizer.transform(df['tokens'])

print(bow.shape)
```

```
(1999, 4271)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer = TfidfVectorizer(analyzer=lambda x: x)

vectorizer.fit(df['Stem'])


tfidf = vectorizer.transform(df['Stem'])

tfidf_array = tfidf.toarray()

all_zeros = (tfidf_array == 0).all()

if not all_zeros:
    print(tfidf_array[0])
```

```
[0.3943827  0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.24444796 0.12987231 0.10011226 0.21470886 0.40884261 0.29023513
 0.12972179 0.23484967 0.34095845 0.         0.15309538 0.18376484
 0.32930567 0.08609939 0.08749485 0.         0.         0.16659404
 0.18357889 0.17297874 0.         0.         0.         0.
 0.         0.        ]
```

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer = TfidfVectorizer(analyzer=lambda x: x)


vectorizer.fit(df['tokens'])


tfidf = vectorizer.transform(df['tokens'])


tfidf_array = tfidf.toarray()


average_tfidf = np.mean(tfidf_array, axis=0)


highest_indices = np.argsort(-average_tfidf)[:10]  # Use negative sign for descending order
lowest_indices = np.argsort(average_tfidf)[:10]

feature_names = vectorizer.get_feature_names_out()


print("Words with the highest average TF-IDF:")
for index in highest_indices:
    print(feature_names[index], average_tfidf[index])


print("\nWords with the lowest average TF-IDF:")
```

```
for index in lowest_indices:
    print(feature_names[index], average_tfidf[index])
```

```
 Words with the highest average TF-IDF:
 australia 0.009983014998891405
 australian 0.00969733014866161
 new 0.008703107457097207
 polic 0.0077360592047481126
 say 0.007540459757782178
 trump 0.006840891998202155
 man 0.006548453421337382
 wa 0.006274671593818188
 charg 0.006028832916829903
 sydney 0.0056424159732095394

 Words with the lowest average TF-IDF:
 nmfc 0.0001527054029533165
 coll 0.0001527054029533165
 melb 0.0001527054029533165
 haw 0.0001527054029533165
 adel 0.0001527054029533165
 syd 0.0001527054029533165
 gcfc 0.0001527054029533165
 gw 0.0001527054029533165
 geel 0.0001527054029533165
 fabio 0.0001613676677950104
```

```
from sklearn.feature_extraction.text import TfidfVectorizer



documents = df['tokens'].apply(lambda x: ' '.join(x))


vectorizer = TfidfVectorizer()

tfidf = vectorizer.fit_transform(documents)


import numpy as np

# Assuming you have computed the TF-IDF representation and stored it in 'tfidf'

# Compute the average TF-IDF values for each feature (word)
average_tfidf = np.mean(tfidf.toarray(), axis=0)

# Get the indices of the words with the highest and lowest average TF-IDF values
highest_indices = np.argsort(-average_tfidf)[:10]  # Use negative sign for descending order
lowest_indices = np.argsort(average_tfidf)[:10]

# Get the feature names (words) from the vectorizer
feature_names = vectorizer.get_feature_names_out()

# Print the words with the highest average TF-IDF values
print("Words with the highest average TF-IDF:")
for index in highest_indices:
    print(feature_names[index], average_tfidf[index])

# Print the words with the lowest average TF-IDF values
print("\nWords with the lowest average TF-IDF:")
for index in lowest_indices:
    print(feature_names[index], average_tfidf[index])
```

```
 Words with the highest average TF-IDF:
 australia 0.009983014998891405
 australian 0.009729510942149733
 new 0.008703107457097207
 polic 0.007736059204748111
 say 0.007555848605072935
 trump 0.006840891998202155
 man 0.006548453421337382
 wa 0.006274671593818188
 charg 0.006028832916829904
 sydney 0.005659788840016151

 Words with the lowest average TF-IDF:
 adel 0.0001527054029533165
 melb 0.0001527054029533165
 haw 0.0001527054029533165
```

```
coll 0.0001527054029533165
gw 0.0001527054029533165
syd 0.0001527054029533165
gcfc 0.0001527054029533165
nmfc 0.0001527054029533165
geel 0.0001527054029533165
fabio 0.00016136766779501044
```

3. Plagiarism checker

```
file = open('/content/drive/MyDrive/Colab/A1.txt', 'r')
A1 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/Asource.txt', 'r')
A0 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/B1.txt', 'r')
B1 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/Bsource.txt', 'r')
B0 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/C1.txt', 'r')
C1 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/Csource.txt', 'r')
C0 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/D1.txt', 'r')
D1 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/D2.txt', 'r')
D2 = file.readlines()[0]

file = open('/content/drive/MyDrive/Colab/Dsource.txt', 'r')
D0 = file.readlines()[0]


C0
```

```
'Descartes has been heralded as the first modern philosopher. He is famous for having m
ade an important connection between geometry and algebra, which allowed for the solving
of geometrical problems by way of algebraic equations. He is also famous for having pro
moted a new conception of matter, which allowed for the accounting of physical phenomen
a by way of mechanical explanations. However, he is most famous for having written a re
```

```
alldata = [A0, A1, B0, B1, C0, C1, D0, D1, D2]


#TODO: Compute tf-idf for all documents
from sklearn.feature_extraction.text import TfidfVectorizer

tfvect = TfidfVectorizer()
tfvect.fit(alldata)


tfidf = tfvect.fit_transform(alldata).toarray()
# Compute the TF-IDF representation for each document
TFIDFA = tfvect.transform([A0, A1]).toarray()
TFIDFB = tfvect.transform([B0, B1]).toarray()
TFIDFC = tfvect.transform([C0, C1]).toarray()
TFIDFD = tfvect.transform([D0, D1, D2]).toarray()




from sklearn.metrics.pairwise import cosine_similarity

# Compute pairwise similarity for document A
similarityAA = cosine_similarity(TFIDFA, TFIDFA)

# Compute pairwise similarity for document B
similarityBB = cosine_similarity(TFIDFB, TFIDFB)

# Compute pairwise similarity for document C
similarityCC = cosine_similarity(TFIDFC, TFIDFC)
```

```
# Compute pairwise similarity for document D
similarityDD = cosine_similarity(TFIDFD, TFIDFD)

# Compute pairwise similarity for all documents
similarityAll = cosine_similarity(tfidf, tfidf)


similarityAA
```

```
array([[1.        , 0.81898863],
       [0.81898863, 1.        ]])
```

```
similarityBB
```

```
array([[1.        , 0.63747903],
       [0.63747903, 1.        ]])
```

```
similarityCC
```

```
array([[1.        , 0.85723864],
       [0.85723864, 1.        ]])
```

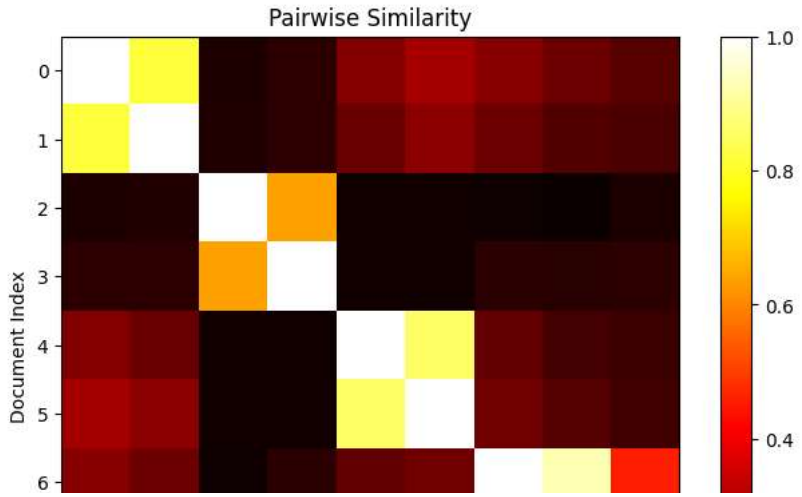
```
similarityDD
```

```
array([[1.        , 0.92754461, 0.45775827],
       [0.92754461, 1.        , 0.47179638],
       [0.45775827, 0.47179638, 1.        ]])
```

```
similarityAll
```

```
array([[1.        , 0.81898863, 0.10748497, 0.12736224, 0.24491604,
        0.29501242, 0.2530779 , 0.2146434 , 0.18585492],
       [0.81898863, 1.        , 0.10940658, 0.12644471, 0.21239125,
        0.25583485, 0.214717  , 0.17937458, 0.16807197],
       [0.10748497, 0.10940658, 1.        , 0.63747903, 0.08804551,
        0.09372089, 0.08509508, 0.07946359, 0.10627147],
       [0.12736224, 0.12644471, 0.63747903, 1.        , 0.08730757,
        0.08893992, 0.12527568, 0.1207279 , 0.12910021],
       [0.24491604, 0.21239125, 0.08804551, 0.08730757, 1.        ,
        0.85723864, 0.20151171, 0.16200275, 0.14447728],
       [0.29501242, 0.25583485, 0.09372089, 0.08893992, 0.85723864,
        1.        , 0.22326634, 0.1806631 , 0.1558733 ],
       [0.2530779 , 0.214717  , 0.08509508, 0.12527568, 0.20151171,
        0.22326634, 1.        , 0.92754461, 0.45775827],
       [0.2146434 , 0.17937458, 0.07946359, 0.1207279 , 0.16200275,
        0.1806631 , 0.92754461, 1.        , 0.47179638],
       [0.18585492, 0.16807197, 0.10627147, 0.12910021, 0.14447728,
        0.1558733 , 0.45775827, 0.47179638, 1.        ]])
```

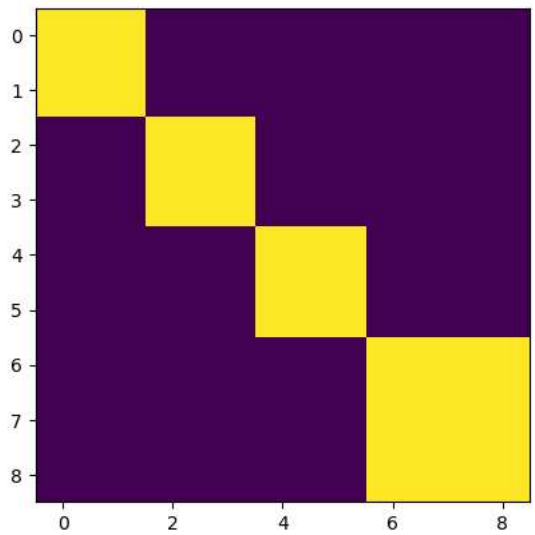
```
import matplotlib.pyplot as plt
pred_plagiarism = similarityAll>0.2
# Function to plot the pairwise similarity matrix
def plot_similarity(similarity_matrix):
    plt.figure(figsize=(8, 6))
    plt.imshow(similarity_matrix, cmap='hot', interpolation='nearest')
    plt.colorbar()
    plt.title('Pairwise Similarity')
    plt.xlabel('Document Index')
    plt.ylabel('Document Index')
    plt.show()



# Plot the pairwise similarity for all documents
plot_similarity(similarityAll)
```




```
import numpy as np

# Initialize the true labels matrix
real_plagiarism = np.zeros((9, 9))

# Set the true labels for plagiarized pairs
real_plagiarism[0:2, 0:2] = 1  # A0 and A1 are plagiarized
real_plagiarism[2:4, 2:4] = 1  # B0 and B1 are plagiarized
real_plagiarism[4:6, 4:6] = 1  # C0 and C1 are plagiarized
real_plagiarism[6:9, 6:9] = 1  # D0, D1, and D2 are plagiarized

# Print the true labels matrix
print(real_plagiarism)
```

```
[[1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

```
import matplotlib.pyplot as plt
plt.imshow(real_plagiarism)
plt.show()
```




```
from sklearn.metrics import accuracy_score

# Convert similarity matrix to binary predictions
threshold = 0.5
```

```
binary_predictions_All = (similarityAll > threshold).astype(int).flatten()

# Convert true labels matrix to binary labels
true_labels = real_plagiarism.flatten()

# Compute accuracy score
accuracy_All = accuracy_score(true_labels, binary_predictions_All)

print("Accuracy for All Documents:", accuracy_All)
```

```
Accuracy for All Documents: 0.9506172839506173
```

III. Text Classification

```
# Import NLTK and all the needed libraries
import nltk
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer



# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Colab/spam.csv', encoding='latin-1')

# Display the first few rows of the dataframe
print(df.head())
```

```
  Class                                            Message
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Class    5572 non-null   object
 1   Message  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Define the preprocess_text function
def preprocess_text(text):
    # Tokenization
    tokens = word_tokenize(text)

    # Lowercasing
    tokens_lower = [token.lower() for token in tokens]

    # Stopword Removal
    stop_words = set(stopwords.words('english'))  # Specify 'latin-1' stopwords
    tokens_no_stopwords = [token for token in tokens_lower if token not in stop_words]

    # Stemming
    stemmer = PorterStemmer()
    tokens_stemmed = [stemmer.stem(token) for token in tokens_no_stopwords]

    # Return preprocessed tokens as a string
```

```
    return ' '.join(tokens_stemmed)

df['preprocessed_text'] = df['Message'].apply(preprocess_text)
```

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

```
df['preprocessed_text']
```

```
0       go jurong point , crazi .. avail bugi n great ...
1                           ok lar ... joke wif u oni ...
2       free entri 2 wkli comp win fa cup final tkt 21...
3             u dun say earli hor ... u c alreadi say ...
4              nah n't think goe usf , live around though
                              ...
5567    2nd time tri 2 contact u. u ï¿½750 pound prize...
5568                         ï¿½_ b go esplanad fr home ?
5569                        piti , * mood . ... suggest ?
5570    guy bitch act like 'd interest buy someth els ...
5571                                    rofl . true name
Name: preprocessed_text, Length: 5572, dtype: object
```

```
from sklearn.feature_extraction.text import CountVectorizer



vectorizer = CountVectorizer()

bow = vectorizer.fit_transform(df['preprocessed_text'])

print( bow.shape)
```

```
(5572, 7495)
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer



# Get the feature names (words) from the vectorizer
feature_names = vectorizer.get_feature_names_out()

# Create a dataframe from the BOW matrix and feature names
bow_df = pd.DataFrame(bow.toarray(), columns=feature_names)

bow_df
```

	00	000	000pe	008704050406	0089	0121	01223585236	01223585334	0125698789	02	...	½t	½te
0	0	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5567	0	0	0	0	0	0	0	0	0	0	...	0	0
5568	0	0	0	0	0	0	0	0	0	0	...	0	0
5569	0	0	0	0	0	0	0	0	0	0	...	0	0
5570	0	0	0	0	0	0	0	0	0	0	...	0	0
5571	0	0	0	0	0	0	0	0	0	0	...	0	0

5572 rows × 7495 columns

IV. Topic Modelling

```
import pandas as pd
# Load the dataset
df = pd.read_csv('/content/random_headlines.csv')
```

```
# Display the first few rows of the dataframe
print(df.head())
```

```
   publish_date                          headline_text
0      20120305  ute driver hurt in intersection crash
1      20081128           6yo dies in cycling accident
2      20090325          bumper olive harvest expected
3      20100201     replica replaces northernmost sign
4      20080225           woods targets perfect season
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   publish_date   20000 non-null  int64 
 1   headline_text  20000 non-null  object
dtypes: int64(1), object(1)
memory usage: 312.6+ KB
```

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string
```

```
df['lowercase'] = df['headline_text'].str.lower()
df['tokens'] = df['lowercase'].apply(word_tokenize)
df['no_punctuation'] = df['tokens'].apply(lambda tokens: [token for token in tokens if token not in string.punctuation])
stopwords_set = set(stopwords.words('english'))
df['no_stopwords'] = df['no_punctuation'].apply(lambda tokens: [token for token in tokens if token not in stopwords_set])
stemmer = PorterStemmer()
df['stemmed'] = df['no_stopwords'].apply(lambda tokens: [stemmer.stem(token) for token in tokens])
```

```
df['stemmed']
```

```
0                    [ute, driver, hurt, intersect, crash]
1                              [6yo, die, cycl, accid]
2                        [bumper, oliv, harvest, expect]
3                     [replica, replac, northernmost, sign]
4                          [wood, target, perfect, season]
                              ...
19995                [judg, attack, walkinshaw, run, arrow]
19996           [polish, govt, collaps, elect, held, next]
19997                              [drum, friday, may, 29]
19998            [winterbottom, bathurst, provision, pole]
19999    [pull, pork, pawpaw, salad, local, success, st...
Name: stemmed, Length: 20000, dtype: object
```

```
!pip install gensim
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.24.3)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.10.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim) (6.3.0)
```

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import string
from gensim.corpora import Dictionary



lemmatizer = WordNetLemmatizer()
df['lemmatized'] = df['no_stopwords'].apply(lambda tokens: [lemmatizer.lemmatize(token) for token in tokens])



# Create a dictionary of the tokens
dictionary = Dictionary(df['stemmed'])

# Filter out rare and common tokens
dictionary.filter_extremes(no_below=5, no_above=0.5)

# Convert each headline to its BOW representation
df['bow'] = df['stemmed'].apply(lambda tokens: dictionary.doc2bow(tokens))



df['bow']
```

```
0                         [(0, 1), (1, 1), (2, 1), (3, 1)]
1                                 [(4, 1), (5, 1), (6, 1)]
2                                 [(7, 1), (8, 1), (9, 1)]
3                                       [(10, 1), (11, 1)]
4                     [(12, 1), (13, 1), (14, 1), (15, 1)]
                                ...
19995                          [(94, 1), (612, 1), (791, 1)]
19996    [(306, 1), (502, 1), (1125, 1), (1279, 1), (16...
19997            [(122, 1), (362, 1), (435, 1), (1929, 1)]
19998                                 [(129, 1), (2498, 1)]
19999    [(767, 1), (1327, 1), (1500, 1), (1927, 1), (2...
Name: bow, Length: 20000, dtype: object
```

```
from gensim.corpora import Dictionary
from gensim.models import TfidfModel


# Create a dictionary of the tokens
dictionary = Dictionary(df['stemmed'])

# Filter out rare and common tokens
dictionary.filter_extremes(no_below=5, no_above=0.5)

# Convert each headline to its BOW representation
df['bow'] = df['stemmed'].apply(lambda tokens: dictionary.doc2bow(tokens))

# Create a list of BOW representations
bow_corpus = df['bow'].tolist()

# Create the TF-IDF model
tfidf_model = TfidfModel(bow_corpus)

# Convert each BOW representation to TF-IDF representation
df['tfidf'] = df['bow'].apply(lambda bow: tfidf_model[bow])



df['tfidf']
```

```
0        [(0, 0.38380472505678387), (1, 0.4408152091819...
1        [(4, 0.565040562691219), (5, 0.678282623325040...
2        [(7, 0.4778523954688469), (8, 0.56145873585743...
3        [(10, 0.7596694993542321), (11, 0.650309350809...
4        [(12, 0.5937887452228604), (13, 0.432499245079...
                               ...
19995    [(94, 0.6788936377183876), (612, 0.47999246040...
19996    [(306, 0.3169859350661769), (502, 0.2621260609...
19997    [(122, 0.37474224836119696), (362, 0.485431117...
19998    [(129, 0.6582992963848873), (2498, 0.752756292...
19999    [(767, 0.43161038088504416), (1327, 0.42560245...
Name: tfidf, Length: 20000, dtype: object
```

```
from gensim.models import  LsiModel
```

```
# Convert each BOW representation to TF-IDF representation
df['tfidf'] = df['bow'].apply(lambda bow: tfidf_model[bow])

# Create the LSA model
lsa_model = LsiModel(df['tfidf'], num_topics=10, id2word=dictionary)

# Convert each TF-IDF representation to LSA representation
df['lsa'] = df['tfidf'].apply(lambda tfidf: lsa_model[tfidf])



df['lsa']
```

```
0        [(0, 0.004248251243656648), (1, -0.07490573060...
1        [(0, 0.0014078164239774565), (1, -0.0606031777...
2        [(0, 0.0011398116481298876), (1, -0.0086400486...
3        [(0, 0.0006727772968257658), (1, -0.0125232086...
4        [(0, 0.0004732836426305638), (1, -0.0140344296...
                              ...
19995    [(0, 0.0023562327759749234), (1, -0.0749580930...
19996    [(0, 0.002296970742186259), (1, -0.03796353044...
19997    [(0, 0.002323511716728376), (1, -0.01207133007...
19998    [(0, 2.081406546614256e-05), (1, -0.0020924029...
19999    [(0, 0.0005920407500201154), (1, -0.0108886780...
Name: lsa, Length: 20000, dtype: object
```

```
# Print the most significant words for each topic
for topic_id, topic in lsa_model.show_topics(num_topics=10, num_words=4, formatted=False):
    words = [f"{word}: {weight:.3f}" for word, weight in topic]
    topic_str = f"({topic_id}, '{', '.join(words)}')"
    print(topic_str)
```

```
(0, 'interview: 0.989, michael: 0.058, extend: 0.042, andrew: 0.037')
(1, 'man: -0.463, polic: -0.384, charg: -0.325, court: -0.160')
(2, 'man: -0.440, charg: -0.313, plan: 0.243, new: 0.215')
(3, 'polic: -0.758, man: 0.230, charg: 0.219, second: 0.143')
(4, 'second: 0.445, 90: 0.404, abc: 0.386, news: 0.344')
(5, 'new: 0.808, plan: -0.216, fire: -0.158, council: -0.156')
(6, 'fire: -0.420, crash: -0.315, kill: -0.299, plan: 0.292')
(7, 'fire: -0.731, crash: 0.261, kill: 0.228, charg: -0.227')
(8, 'win: -0.441, plan: 0.389, new: 0.298, court: -0.285')
(9, 'court: 0.569, charg: -0.519, face: 0.273, win: -0.175')
```

```
from gensim.models import  LdaModel


lda_model = LdaModel(df['tfidf'], num_topics=10, id2word=dictionary)

# Convert each TF-IDF representation to LDA representation
df['lda'] = df['tfidf'].apply(lambda tfidf: lda_model[tfidf])
df['lda']
```

```
0        [(0, 0.03368945), (1, 0.69677943), (2, 0.03369...
1        [(0, 0.036864247), (1, 0.036888707), (2, 0.036...
2        [(0, 0.0368385), (1, 0.036857087), (2, 0.03683...
3        [(0, 0.041502547), (1, 0.6263373), (2, 0.04150...
4        [(0, 0.033538155), (1, 0.033552643), (2, 0.033...
                              ...
19995    [(0, 0.2677625), (1, 0.036852058), (2, 0.03685...
19996    [(0, 0.029507797), (1, 0.029510105), (2, 0.227...
19997    [(0, 0.033685252), (1, 0.033683848), (2, 0.033...
19998    [(0, 0.3535773), (1, 0.041485377), (2, 0.04148...
19999    [(0, 0.031085761), (1, 0.031085763), (2, 0.031...
Name: lda, Length: 20000, dtype: object
```

```
# Print the most frequent words for each topic
for topic_id, topic in lda_model.show_topics(num_topics=10, num_words=3, formatted=False):
    words = [f"{word}: {weight:.3f}" for word, weight in topic]
    topic_str = f"({topic_id}, '{', '.join(words)}')"
    print(topic_str)
```

```
(0, 'man: 0.019, polic: 0.016, charg: 0.015')
(1, 'crash: 0.009, die: 0.009, drought: 0.009')
(2, 'win: 0.011, perth: 0.009, world: 0.008')
(3, 'countri: 0.010, hour: 0.008, rail: 0.008')
```

```
(4, 'interview: 0.035, news: 0.015, fatal: 0.009')
(5, 'talk: 0.011, abc: 0.010, start: 0.010')
(6, 'plan: 0.009, question: 0.008, inquiri: 0.006')
(7, 'second: 0.014, weather: 0.012, kill: 0.010')
(8, 'futur: 0.008, make: 0.008, union: 0.007')
(9, 'brisban: 0.010, clash: 0.008, famili: 0.008')
```

```
!pip install pyLDAvis
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyLDAvis in /usr/local/lib/python3.10/dist-packages (3.4.1)
Requirement already satisfied: numpy>=1.24.2 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.24.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.10.1)
Requirement already satisfied: pandas>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (2.0.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.2.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (3.1.2)
Requirement already satisfied: numexpr in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (2.8.4)
Requirement already satisfied: funcy in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (2.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (1.2.2)
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (4.3.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from pyLDAvis) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis) (2022.7.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.0.0->pyLDAvis) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->pyLDAvis) (3.1
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim->pyLDAvis) (6.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->pyLDAvis) (2.1.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=2.0.0->pyLDAvis
```

```
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis
```

```
corpus = df['tfidf']
id2word = dictionary
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
!pip install joblib
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (1.2.0)
```

```
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis
from joblib import Memory
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
cache_dir = './cache'
memory = Memory(cache_dir, verbose=0)
```

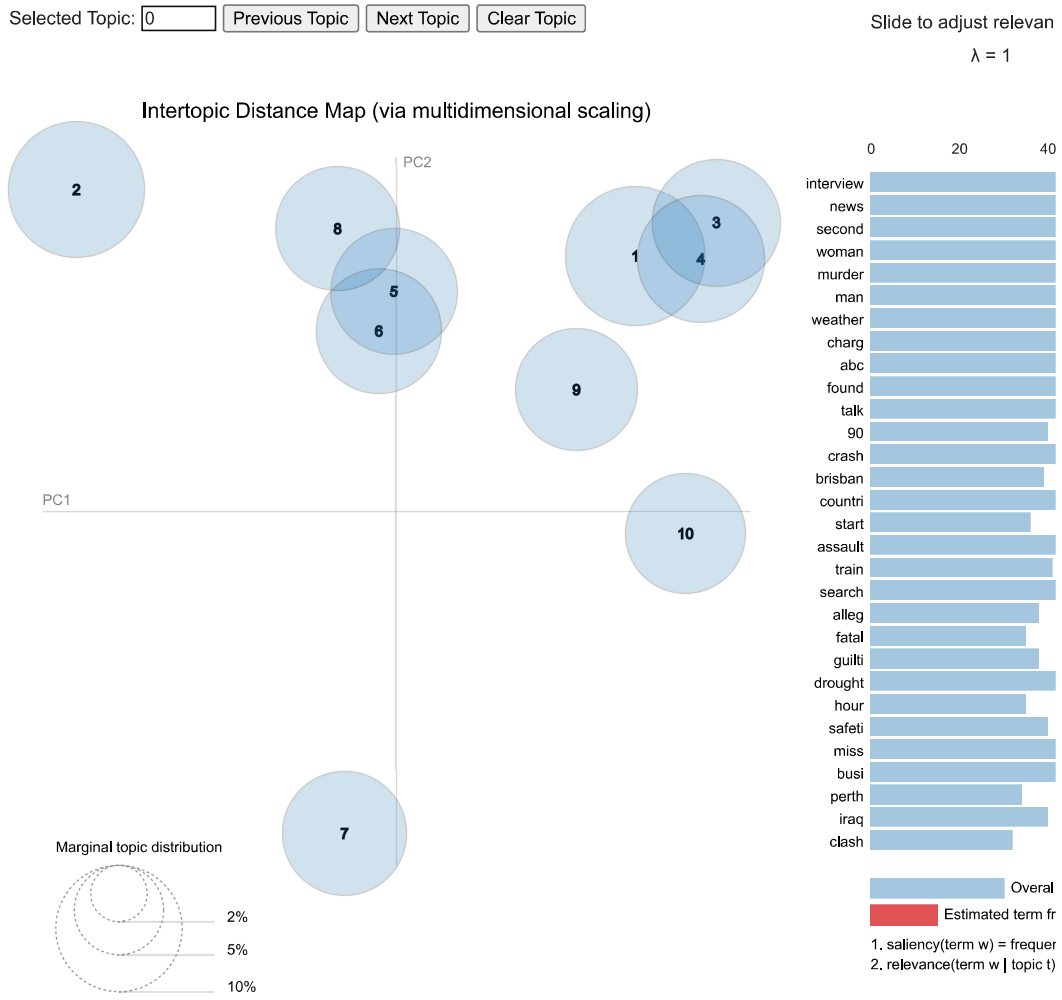
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
@memory.cache
def compute_vis_data():
    return gensimvis.prepare(lda_model, corpus, id2word)

vis_data = compute_vis_data()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
pyLDAvis.display(vis_data)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_asyr
  and should_run_async(code)
```



V. Named Entity Recognition

```
df = "/content/drive/MyDrive/Colab/ada_lovelace.txt"
```

```
def clean_file(filename):
    with open(filename, 'r') as file:
        contents = file.read()

    redacted_contents = contents.replace("Ada Lovelace", "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

clean_file(df)
```

```
import spacy
```

```
def identify_entities(filename):
    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file:
        contents = file.read()

    doc = nlp(contents)

    for entity in doc.ents:
        print(entity.text, entity.label_)



identify_entities(df)
```

```
 Augusta Ada King PERSON
 Countess PERSON
 Lovelace PERSON
 Byron ORG
 10 December 1815 DATE
 27 November 1852 DATE
 English LANGUAGE
 Charles Babbage's ORG
 the Analytical Engine ORG
 first ORDINAL
 first ORDINAL
 first ORDINAL
 one CARDINAL
 first ORDINAL
 Lovelace PERSON
 Mary Somerville PERSON
 Charles Babbage PERSON
 1833 DATE
 Somerville GPE
 many years DATE
 Andrew Crosse PERSON
 David Brewster PERSON
 Charles Wheatstone PERSON
 Michael Faraday PERSON
 Charles Dickens PERSON
```

```
import spacy
from spacy import displacy
from IPython.display import display

def visualize_entities(filename):

    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file:
        contents = file.read()

    doc = nlp(contents)

    displacy.render(doc, style="ent", jupyter=True)



visualize_entities(df)
```

Augusta Ada King **PERSON** , Countess **PERSON** of Lovelace **PERSON** (née Byron **ORG** ; 10 December 1815 **DATE** – 27 November 1852 **DATE**) was an English **LANGUAGE** mathematician and writer, chiefly known for her work on Charles Babbage's **ORG** proposed mechanical general-purpose computer, the Analytical Engine **ORG** . She was the first **ORDINAL** to recognise that the machine had applications beyond pure calculation, and published the first **ORDINAL** algorithm intended to be carried out by such a machine. As a result, she is sometimes regarded as the first **ORDINAL** to recognise the full potential of a "computing machine" and one **CARDINAL** of the first **ORDINAL** computer programmers.

Lovelace **PERSON** became close friends with her tutor Mary Somerville **PERSON** , who introduced her to Charles

```
import spacy

def replace_name_by_redacted(filename):
    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file:
        contents = file.read()

    doc = nlp(contents)

    redacted_contents = contents
    for entity in doc.ents:
        if entity.label_ == "PERSON":
            redacted_contents = redacted_contents.replace(entity.text, "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

replace_name_by_redacted(df)
```

```
import spacy

def make_doc_GDPR_compliant(filename):

    nlp = spacy.load("en_core_web_sm")

    with open(filename, 'r') as file:
        contents = file.read()

    doc = nlp(contents)

    redacted_contents = contents
    for entity in doc.ents:
        if entity.label_ == "PERSON":
            redacted_contents = redacted_contents.replace(entity.text, "[REDACTED]")

    with open(filename, 'w') as file:
        file.write(redacted_contents)

make_doc_GDPR_compliant(df)
```

VI. Exercise

```
df = pd.read_csv('/content/drive/MyDrive/Colab/job-market.csv')
```

```
df.fillna(0)
```

	Id	Title	Company	Date	Location	Area	Classification
0	37404348.0	Casual Stock Replenisher	Aldi Stores	2018-10-07T00:00:00.000Z	Sydney	North West & Hills District	Retail & Consumer Products
1	37404337.0	Casual Stock Replenisher	Aldi Stores	2018-10-07T00:00:00.000Z	Richmond & Hawkesbury	0	Retail & Consumer Products
2	37404356.0	RETAIL SALES SUPERSTARS and STYLISTS Wanted - ...	LB Creative Pty Ltd	2018-10-07T00:00:00.000Z	Brisbane	CBD & Inner Suburbs	Retail & Consumer Products

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

# Step 1: Filter the jobs for the IT sector only
df['Title'] = df['Title'].fillna('')
IT_df = df[df['Title'].str.contains('IT', case=False)]
```

Specialist, Ni... Banking

```
job_descriptions = IT_df['Classification'].tolist()
```

```
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
X = vectorizer.fit_transform(job_descriptions)
# Train a logistic regression model
model = LogisticRegression()
model.fit(X, IT_df['Title'])

# Get feature names and their corresponding coefficients
feature_names = vectorizer.get_feature_names_out()
coefficients = model.coef_[0]

# Sort feature names based on coefficients
top_keywords = sorted(zip(coefficients, feature_names), reverse=True)[:20]

# Print the top 20 important keywords
print("Top 20 important keywords:")
for coef, keyword in top_keywords:
    print(keyword)
```

```
Top 20 important keywords:
retail
products
consumer
management
general
ceo
strategy
consulting
science
farming
conservation
animals
superannuation
insurance
media
arts
advertising
financial
banking
real
```

```
keyword = 'python'

query = vectorizer.transform([keyword])

similarity_scores = model.predict_proba(query)[0]
```

```
ranked_jobs = sorted(zip(similarity_scores, IT_df['Title'], job_descriptions), reverse=True)

print("\nTop 5 job descriptions most similar to the query:")
for score, job_title, job_description in ranked_jobs[:5]:
    print("Job Title:", job_title)
    print("Similarity Score:", score)
    print("Job Description:", job_description)
    print()
```

```
 Top 5 job descriptions most similar to the query:
 Job Title: Credit Controller - Temporary Position
 Similarity Score: 0.004172072785443803
 Job Description: Accounting

 Job Title: Wait Staff
 Similarity Score: 0.0033997001440222003
 Job Description: Hospitality & Tourism

 Job Title: Solution Architect (IAM)
 Similarity Score: 0.003043823447949755
 Job Description: Information & Communication Technology

 Job Title: Recruitment Consultant
 Similarity Score: 0.0027987719273321284
 Job Description: Human Resources & Recruitment

 Job Title: IT Security Architect
 Similarity Score: 0.002747864116015302
 Job Description: Information & Communication Technology
```

2

```
def extract_ngrams(sequence, n):
    ngrams = []
    sequence_length = len(sequence)
    for i in range(sequence_length - n + 1):
        ngram = sequence[i:i+n]
        ngrams.append(ngram)
    return ngrams


sentence = "I like deadline and want to immerse myself in deadline."

# Extract word tri-grams
words = sentence.split()
word_trigrams = extract_ngrams(words, 3)

print("Word Tri-grams:")
for trigram in word_trigrams:
    print(trigram)

# Extract letter tri-grams
letters = list(sentence.replace(" ", ""))
letter_trigrams = extract_ngrams(letters, 3)

print("\nLetter Tri-grams:")
for trigram in letter_trigrams:
    print(trigram)
```

```
 Word Tri-grams:
 ['I', 'like', 'deadline']
 ['like', 'deadline', 'and']
 ['deadline', 'and', 'want']
 ['and', 'want', 'to']
 ['want', 'to', 'immerse']
 ['to', 'immerse', 'myself']
 ['immerse', 'myself', 'in']
 ['myself', 'in', 'deadline.']

 Letter Tri-grams:
 ['I', 'l', 'i']
 ['l', 'i', 'k']
 ['i', 'k', 'e']
 ['k', 'e', 'd']
 ['e', 'd', 'e']
```

```
['d', 'e', 'a']
['e', 'a', 'd']
['a', 'd', 'l']
['d', 'l', 'i']
['l', 'i', 'n']
['i', 'n', 'e']
['n', 'e', 'a']
['e', 'a', 'n']
['a', 'n', 'd']
['n', 'd', 'w']
['d', 'w', 'a']
['w', 'a', 'n']
['a', 'n', 't']
['n', 't', 't']
['t', 't', 'o']
['t', 'o', 'i']
['o', 'i', 'm']
['i', 'm', 'm']
['m', 'm', 'e']
['m', 'e', 'r']
['e', 'r', 's']
['r', 's', 'e']
['s', 'e', 'm']
['e', 'm', 'y']
['m', 'y', 's']
['y', 's', 'e']
['s', 'e', 'l']
['e', 'l', 'f']
['l', 'f', 'i']
['f', 'i', 'n']
['i', 'n', 'd']
['n', 'd', 'e']
['d', 'e', 'a']
['e', 'a', 'd']
['a', 'd', 'l']
['d', 'l', 'i']
['l', 'i', 'n']
['i', 'n', 'e']
['n', 'e', '.']
```

3

```
import random

def modify_phrase(phrase):
    words = phrase.split()
    modified_words = []

    for word in words:
        if len(word) <= 4:
            modified_words.append(word)
        else:
            first_letter = word[0]
            last_letter = word[-1]
            middle_letters = list(word[1:-1])
            random.shuffle(middle_letters)
            modified_word = first_letter + ''.join(middle_letters) + last_letter
            modified_words.append(modified_word)

    modified_phrase = ' '.join(modified_words)
    return modified_phrase

# Example phrase
phrase = "I couldn't believe that I could completely understand what I was reading: the astounding power of the human mind"

modified_phrase = modify_phrase(phrase)

print("Original phrase:")
print(phrase)
print()
print("Modified phrase:")
print(modified_phrase)
```

```
Original phrase:
I couldn't believe that I could completely understand what I was reading: the astounding power of the human mind

Modified phrase:
I cndolu't bveilee that I cloud celtlmopey uadnsenrtd what I was rgeandi: the auotnsidng pweor of the haumn mind
```

4

```
alice = "/content/drive/MyDrive/Colab/alice.txt"
```

```
import nltk

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
import nltk

# Read the input file
with open(alice, 'r') as file:
    text = file.read()

# Tokenize the text into sentences
sentences = nltk.sent_tokenize(text)

# Perform POS tagging on each sentence
tagged_sentences = []
for sentence in sentences:
    tagged_sentence = nltk.pos_tag(nltk.word_tokenize(sentence))
    tagged_sentences.append(tagged_sentence)

# Save the POS tagged output to a separate file
output_file = 'alice_pos_tagged.txt'
with open(output_file, 'w') as file:
    for tagged_sentence in tagged_sentences:
        tagged_text = ' '.join([f"{word}/{tag}" for word, tag in tagged_sentence])
        file.write(tagged_text + '\n')

print(f"POS tagged output saved to '{output_file}'.")
```

```
POS tagged output saved to 'alice_pos_tagged.txt'.
```

```
with open('alice_pos_tagged.txt', 'r') as file:
    pos_tagged_text = file.read()

# Print the contents of the POS tagged file
print("POS tagged text:")
print(pos_tagged_text)
```

```
POS tagged text:
ALICE/NNP 'S/POS ADVENTURES/NNP IN/NNP WONDERLAND/NNP Lewis/NNP Carroll/NNP THE/NNP MILLENNIUM/NNP FULCRUM/NNP EDITION/NNP 3.0/CD CHA
Down/IN the/DT Rabbit-Hole/JJ Alice/NNP was/VBD beginning/VBG to/TO get/VB very/RB tired/JJ of/IN sitting/VBG by/IN her/PRP$ sister/N
So/IN she/PRP was/VBD considering/VBG in/IN her/PRP$ own/JJ mind/NN (/( as/RB well/RB as/IN she/PRP could/MD ,/, for/IN the/DT hot/JJ
There/EX was/VBD nothing/NN so/RB VERY/RB remarkable/JJ in/IN that/DT ;/: nor/CC did/VBD Alice/NNP think/VB it/PRP so/RB VERY/RB much
Oh/UH dear/NN !/.
I/PRP shall/MD be/VB late/RB !/. '/''
(/( when/WRB she/PRP thought/VBD it/PRP over/IN afterwards/NNS ,/, it/PRP occurred/VBD to/TO her/PRP$ that/IN she/PRP ought/MD to/TO
In/IN another/DT moment/NN down/RP went/VBD Alice/NNP after/IN it/PRP ,/, never/RB once/RB considering/VBG how/WRB in/IN the/DT world
The/DT rabbit-hole/JJ went/VBD straight/RB on/IN like/IN a/DT tunnel/NN for/IN some/DT way/NN ,/, and/CC then/RB dipped/VBD suddenly/
Either/CC the/DT well/NN was/VBD very/RB deep/JJ ,/, or/CC she/PRP fell/VBD very/RB slowly/RB ,/, for/IN she/PRP had/VBD plenty/NN of
First/RB ,/, she/PRP tried/VBD to/TO look/VB down/RP and/CC make/VB out/RP what/WP she/PRP was/VBD coming/VBG to/TO ,/, but/CC it/PRP
She/PRP took/VBD down/RP a/DT jar/NN from/IN one/CD of/IN the/DT shelves/NNS as/IN she/PRP passed/VBD ;/: it/PRP was/VBD labelled/VBN
'Well/RB !/. '/''
thought/VBN Alice/NNP to/TO herself/VB ,/, 'after/FW such/PDT a/DT fall/NN as/IN this/DT ,/, I/PRP shall/MD think/VB nothing/NN of/IN
How/WRB brave/VBP they/PRP 'll/MD all/DT think/VB me/PRP at/IN home/NN !/.
Why/WRB ,/, I/PRP would/MD n't/RB say/VB anything/NN about/IN it/PRP ,/, even/RB if/IN I/PRP fell/VBD off/RP the/DT top/NN of/IN the/
(/( Which/NNP was/VBD very/RB likely/JJ true/JJ ./. )/)
Down/NNP ,/, down/RB ,/, down/RB ./.
Would/MD the/DT fall/NN NEVER/NNP come/VBP to/TO an/DT end/NN !/.
'/POS I/PRP wonder/VBP how/WRB many/JJ miles/NNS I/PRP 've/VBP fallen/VBN by/IN this/DT time/NN ?/. '/''
she/PRP said/VBD aloud/NN ./.
'/POS I/PRP must/MD be/VB getting/VBG somewhere/RB near/IN the/DT centre/NN of/IN the/DT earth/NN ./.
Let/VB me/PRP see/VB :/: that/DT would/MD be/VB four/CD thousand/NN miles/NNS down/RB ,/, I/PRP think/VBP --/: '/POS (/( for/IN ,/, y
(/( Alice/NNP had/VBD no/DT idea/NN what/WP Latitude/NNP was/VBD ,/, or/CC Longitude/NNP either/CC ,/, but/CC thought/VBD they/PRP we
Presently/RB she/PRP began/VBD again/RB ./.
```

```
'/POS I/PRP wonder/VBP if/IN I/PRP shall/MD fall/VB right/RB THROUGH/IN the/DT earth/NN !/.
How/WRB funny/JJ it/PRP 'll/MD seem/VB to/TO come/VB out/RP among/IN the/DT people/NNS that/WDT walk/VBP with/IN their/PRP$ heads/NNS
The/DT Antipathies/NNPS ,/, I/PRP think/VBP --/: '/'' (/( she/PRP was/VBD rather/RB glad/JJ there/EX WAS/NNP no/DT one/NN listening/N
Please/NNP ,/, Ma'am/NNP ,/, is/VBZ this/DT New/NNP Zealand/NNP or/CC Australia/NNP ?/. '/''
(/( and/CC she/PRP tried/VBD to/TO curtsey/VB as/IN she/PRP spoke/VBD --/: fancy/JJ CURTSEYING/NN as/IN you/PRP 're/VBP falling/VBG t
Do/VBP you/PRP think/VB you/PRP could/MD manage/VB it/PRP ?/. )/)
'And/POS what/WP an/DT ignorant/JJ little/JJ girl/NN she/PRP 'll/MD think/VB me/PRP for/IN asking/VBG !/.
No/DT ,/, it/PRP 'll/MD never/RB do/VB to/TO ask/VB :/: perhaps/RB I/PRP shall/MD see/VB it/PRP written/VBN up/RP somewhere/RB ./. '/
Down/NNP ,/, down/RB ,/, down/RB ./.
There/EX was/VBD nothing/NN else/RB to/TO do/VB ,/, so/IN Alice/NNP soon/RB began/VBD talking/VBG again/RB ./.
'Dinah/POS 'll/MD miss/VB me/PRP very/RB much/JJ to-night/NN ,/, I/PRP should/MD think/VB !/. '/''
(/( Dinah/NNP was/VBD the/DT cat/NN ./. )/)
'/POS I/PRP hope/VBP they/PRP 'll/MD remember/VB her/PRP$ saucer/NN of/IN milk/NN at/IN tea-time/NN ./.
Dinah/NNP my/PRP$ dear/NN !/.
I/PRP wish/VBP you/PRP were/VBD down/RB here/RB with/IN me/PRP !/.
There/EX are/VBP no/DT mice/NN in/IN the/DT air/NN ,/, I/PRP 'm/VBP afraid/JJ ,/, but/CC you/PRP might/MD catch/VB a/DT bat/NN ,/, an
But/CC do/VBP cats/NNS eat/VB bats/NNS ,/, I/PRP wonder/VBP ?/. '/''
And/CC here/RB Alice/NNP began/VBD to/TO get/VB rather/RB sleepy/NN ,/, and/CC went/VBD on/IN saying/VBG to/TO herself/VB ,/, in/IN a
Do/VB cats/NNS eat/VB bats/NNS ?/. '/''
and/CC sometimes/RB ,/, 'Do/'' bats/VBZ eat/NN cats/NNS ?/. '/''
for/IN ,/, you/PRP see/VBP ,/, as/IN she/PRP could/MD n't/RB answer/VB either/DT question/NN ,/, it/PRP did/VBD n't/RB much/JJ matter
She/PRP felt/VBD that/IN she/PRP was/VBD dozing/VBG off/RP ,/, and/CC had/VBD just/RB begun/VBN to/TO dream/VB that/IN she/PRP was/VB
when/WRB suddenly/RB ,/, thump/NN !/.
thump/NN !/.
down/IN she/PRP came/VBD upon/IN a/DT heap/NN of/IN sticks/NNS and/CC dry/JJ leaves/NNS ,/, and/CC the/DT fall/NN was/VBD over/RB ./.
Alice/NNP was/VBD not/RB a/DT bit/NN hurt/JJ ,/, and/CC she/PRP jumped/VBD up/RB on/IN to/TO her/PRP$ feet/NNS in/IN a/DT moment/NN :
There/EX was/VBD not/RB a/DT moment/NN to/TO be/VB lost/VBN :/: away/RB went/VBD Alice/NNP like/IN the/DT wind/NN ,/, and/CC was/VBD
She/PRP was/VBD close/RB behind/IN it/PRP when/WRB she/PRP turned/VBD the/DT corner/NN ,/, but/CC the/DT Rabbit/NNP was/VBD no/RB lon
There/EX were/VBD doors/NNS all/DT round/VBP the/DT hall/NN ,/, but/CC they/PRP were/VBD all/DT locked/VBN ;/: and/CC when/WRB Alice/
Suddenly/RB she/PRP came/VBD upon/IN a/DT little/JJ three-legged/JJ table/NN ,/, all/DT made/VBN of/IN solid/JJ glass/NN ;/: there/EX
```