In [ ]:
```
#import modules
import os.path

# NLTK is a leading platform for building Python programs to work with human langua
# It provides easy-to-use interfaces to over 50 corpora and lexical resources such
# along with a suite of text processing libraries for classification, tokenization,
# tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP li
# and an active discussion forum. (source: nltk.org)
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Gensim is a Python library for topic modelling, document indexing and similarity
# Target audience is the natural language processing (NLP) and information retrieva
import gensim
from gensim import corpora
from gensim.models import LsiModel
from gensim.models.coherencemodel import CoherenceModel

# import matplotlib.pyplot as plt
```

In [ ]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\LaptopNUC\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]: True

In [ ]:
```
def preprocess_data(doc_set):
    """
    Input  : document list
    Purpose: preprocess text (tokenize, removing stopwords, and stemming)
    Output : preprocessed text
    """
    # initialize regex tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    # create English stop words list
    en_stop = set(stopwords.words('english'))
    # Create p_stemmer of class PorterStemmer
    p_stemmer = PorterStemmer()
    # list for tokenized documents in loop
    texts = []
    # loop through document list
    for i in doc_set:
        # clean and tokenize document string
        raw = i.lower()
        tokens = tokenizer.tokenize(raw)
        # remove stop words from tokens
        stopped_tokens = [i for i in tokens if not i in en_stop]
        # stem tokens
        stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]
        # add tokens to list
```

```
        texts.append(stemmed_tokens)
    return texts
```

In [ ]:
```
my_corpus = ["cat dog horse",
             "cat horse pig",
             "horse pig cow",
             "baseball golf hockey",
            "golf curling hockey"]
```

In [ ]:
```
docs = preprocess_data(my_corpus)
```

In [ ]:
```
docs
```

Out[ ]:
```
[['cat', 'dog', 'hors'],
 ['cat', 'hors', 'pig'],
 ['hors', 'pig', 'cow'],
 ['basebal', 'golf', 'hockey'],
 ['golf', 'curl', 'hockey']]
```

In [ ]:
```
def prepare_corpus(doc_clean):
    """
    Input  : clean document
    Purpose: create term dictionary of our courpus and Converting list of documents
    Output : term dictionary and Term-Document Matrix
    """
    # Creating the term dictionary of our courpus, where every unique term is assig
    dictionary = corpora.Dictionary(doc_clean)
    # Converting list of documents (corpus) into Term-Document Matrix using diction
    term_doc_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
    # generate LDA model
    return dictionary,term_doc_matrix
```

In [ ]:
```
def create_gensim_lsa_model(doc_clean,number_of_topics,words):
    """
    Input  : clean document, number of topics and number of words associated with e
    Purpose: create LSA model using gensim
    Output : return LSA model
    """
    dictionary,term_doc_matrix=prepare_corpus(doc_clean)
    # generate LSA model
    # Note another traditional term for the same method
    # is "Latent Semantic Indexing" (LSI)
    # gensim happens to use the term "LSI" instead of "LSA"
    lsamodel = LsiModel(term_doc_matrix, num_topics=number_of_topics, id2word = dic
    print(lsamodel.print_topics(num_topics=number_of_topics, num_words=words))
    return lsamodel, term_doc_matrix
```

In [ ]:
```
lsa, tdm = create_gensim_lsa_model(docs, 2, 10)
```

```
[(0, '0.684*"hors" + 0.469*"cat" + 0.469*"pig" + 0.215*"dog" + 0.215*"cow" + 0.000
*"curl" + -0.000*"basebal" + -0.000*"golf" + -0.000*"hockey"'), (1, '0.632*"golf" +
0.632*"hockey" + 0.316*"basebal" + 0.316*"curl" + -0.000*"cow" + 0.000*"cat" + -0.00
0*"dog" + 0.000*"pig" + 0.000*"hors"')]
```

In [ ]:
```
# Terms
[ lsa.id2word[i] for i in range(lsa.num_terms) ]
```

Out[ ]:
```
['cat', 'dog', 'hors', 'pig', 'cow', 'basebal', 'golf', 'hockey', 'curl']
```

In [ ]:
```
# Term representations
U = lsa.projection.u
U
```

Out[ ]:
```
array([[ 4.69190324e-01, -1.38830224e-15],
       [ 2.14620373e-01,  9.03729823e-16],
       [ 6.83810697e-01,  4.36031296e-16],
       [ 4.69190324e-01, -4.26226021e-16],
       [ 2.14620373e-01,  1.84626168e-15],
       [-1.06071413e-16,  3.16227766e-01],
       [-2.29988571e-16,  6.32455532e-01],
       [-2.18948590e-16,  6.32455532e-01],
       [-1.04530946e-16,  3.16227766e-01]])
```

In [ ]:
```
# Documents
docs
```

Out[ ]:
```
[['cat', 'dog', 'hors'],
 ['cat', 'hors', 'pig'],
 ['hors', 'pig', 'cow'],
 ['basebal', 'golf', 'hockey'],
 ['golf', 'curl', 'hockey']]
```

In [ ]:
```
# Document representations
V = gensim.matutils.corpus2dense(lsa[tdm], len(lsa.projection.s)).T / lsa.projectio
V.T
```

Out[ ]:
```
array([[0.54177433, 0.64262054, 0.54177433, 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.70710679, 0.70710679]])
```

In [ ]:
```
# Similarity from "cat" to each document. (Dot product)

(U[0,:] * V).sum(axis=1)
```

Out[ ]:
```
array([ 2.54195274e-01,  3.01511338e-01,  2.54195274e-01, -9.81677937e-16,
       -9.81677937e-16])
```

In [ ]:
```
# Term 'cat' is similar to first three documents. But wait!

docs
```

Out[ ]:
```
[['cat', 'dog', 'hors'],
 ['cat', 'hors', 'pig'],
 ['hors', 'pig', 'cow'],
 ['basebal', 'golf', 'hockey'],
 ['golf', 'curl', 'hockey']]
```

In [ ]:
```
# THIRD DOCUMENT DOESN'T EVEN CONTAIN CAT!!!
```