LDA OF SAMPLE DATA USING BAG OF WORDS

```
import pandas as pd
data = pd.read_excel('/content/drive/MyDrive/LDA-Data.xlsx')
data.head()
```

	News
0	Virat scored century in match
1	BJP won in elections
2	Bumra took 5 wicket in a match
3	Congress form state government

Next steps: Generate code with data New interactive sheet

```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
```

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

```
def nltk_preprocessing_pipeline(text):
    # Initialize NLTK tools
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    # 1. Text Preprocessing
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE) #remove URLs
    text = re.sub(r'<.*?>', '', text) #remove HTML tags
    text = re.sub(r'@\w+', '', text) #remove mentions
    text = re.sub(r'#\w+', '', text) #remove hashtags
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces

    # 2. Word Tokenization
    tokenized_words = word_tokenize(text)

    # 3. Stopword Removal
    filtered_words = [word for word in tokenized_words if word not in stop_words]

    # 4. Lemmatization
    lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]

    # 5. Rejoin words
    clean_summary = ' '.join(lemmatized_words)

    return clean_summary

print("NLTK preprocessing pipeline function created successfully!")
```

```
NLTK preprocessing pipeline function created successfully!
```

```
data['clean_News'] = data['News'].apply(nltk_preprocessing_pipeline)
print("\nComparison of previous clean_summaries and new clean_summaries_pipeline (first 5 rows):")
print(data[['clean_News']].head())
```

```

Comparison of previous clean_summaries and new clean_summaries_pipeline (first 5 rows):
```

```
                    clean_News
0      virat scored century match
1                   bjp election
2       bumra took 5 wicket match
3  congress form state government
```

```
from sklearn.feature_extraction.text import CountVectorizer
# Vectorize the cleaned summaries
count_vectorizer = CountVectorizer(max_df=0.95, min_df=1, stop_words='english')
doc_term_matrix = count_vectorizer.fit_transform(data['clean_News'])
```

```
import pandas as pd
# Get feature (word) names
feature_names = count_vectorizer.get_feature_names_out()
# Convert sparse matrix to DataFrame
bow_df = pd.DataFrame(doc_term_matrix.toarray(),columns=feature_names)
# Display BoW matrix for top 10 documents
bow_top_10 = bow_df.head(10)
print(bow_top_10)
```

```
   bjp  bumra  century  congress  election  form  government  match  scored  \
0    0      0        1         0         0     0           0      1       1   
1    1      0        0         0         1     0           0      0       0   
2    0      1        0         0         0     0           0      1       0   
3    0      0        0         1         0     1           1      0       0   

   state  took  virat  wicket  
0      0     0      1       0  
1      0     0      0       0  
2      0     1      0       1  
3      1     0      0       0  
```

```
from sklearn.decomposition import LatentDirichletAllocation
# Initialize and fit LDA model
num_topics = 2
LDA = LatentDirichletAllocation(n_components=num_topics, random_state=42)
LDA.fit(doc_term_matrix)
```

```
▾            LatentDirichletAllocation            i ?

LatentDirichletAllocation(n_components=2, random_state=42)
```

```
def display_topics(model, feature_names, num_top_words):
    for topic_idx in range(len(model.components_)):
        print(f"\nTopic {topic_idx}:")
        # Get word weights for this topic
        topic_weights = model.components_[topic_idx]
        # Get indices of words sorted by weight (descending)
        sorted_indices = topic_weights.argsort()[::-1]
        # Take top N words
        top_indices = sorted_indices[:num_top_words]
        # Print top words
        for idx in top_indices:
            print(feature_names[idx], end=" ")
        print()
```

```
# Display top words for each topic
num_top_words = 10
print(f"\nTop {num_top_words} words per topic:")
display_topics(LDA, count_vectorizer.get_feature_names_out(), num_top_words)
```

```

Top 10 words per topic:

Topic 0:
form government congress state election bjp match wicket bumra took 

Topic 1:
match virat century scored took bumra wicket bjp election state 
```

```
# Assign topics to each document
document_topics = LDA.transform(doc_term_matrix)
data['topic'] = document_topics.argmax(axis=1)
```

```
print("\nDataFrame with assigned topics (first 5 rows):")
print(data[['clean_News', 'topic']].head())
```

```
DataFrame with assigned topics (first 5 rows):
                       clean_News  topic
0      virat scored century match      1
1                    bjp election      0
2       bumra took 5 wicket match      1
3  congress form state government      0
```

NMF OF SAMPLE DATA USING BAG OF WORDS

```
from sklearn.decomposition import NMF

# Initialize NMF model
num_topics = 2
nmf_model = NMF(n_components=num_topics, random_state=42)

# Fit the NMF model to the document-term matrix
nmf_model.fit(doc_term_matrix)
print("NMF model initialized and fitted successfully.")
```

```
NMF model initialized and fitted successfully.
```

```
print(f"\nTop {num_top_words} words per topic (NMF):")
display_topics(nmf_model, count_vectorizer.get_feature_names_out(), num_top_words)
```

```
Top 10 words per topic (NMF):

Topic 0:
match virat took scored wicket bumra century bjp election state

Topic 1:
state form congress government election bjp took virat wicket scored
```

```
# Assign topics to each document
document_topics = nmf_model.transform(doc_term_matrix)
data['topic'] = document_topics.argmax(axis=1)

print("\nDataFrame with assigned topics (first 5 rows):")
print(data[['clean_News', 'topic']].head())
```

```
DataFrame with assigned topics (first 5 rows):
                       clean_News  topic
0      virat scored century match      0
1                    bjp election      1
2       bumra took 5 wicket match      0
3  congress form state government      1
```

LDA OF KAGGLE DATASET USING BAG OF WORDS

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/arxiv_data.csv.zip', engine='python', nrows=1000)
display(df.head())
```

	titles	summaries	terms
0	Survey on Semantic Stereo Matching / Semantic ...	Stereo matching is one of the widely used tech...	['cs.CV', 'cs.LG']
1	FUTURE-AI: Guiding Principles and Consensus Re...	The recent advancements in artificial intellig...	['cs.CV', 'cs.AI', 'cs.LG']
2	Enforcing Mutual Consistency of Hard Regions f...	In this paper, we proposed a novel mutual cons...	['cs.CV', 'cs.AI']
3	Parameter Decoupling Strategy for Semi-supervi...	Consistency training has proven to be an advan...	['cs.CV']
4	Background-Foreground Segmentation for Interio...	To ensure safety in automated driving, the cor...	['cs.CV', 'cs.LG']

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

```
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
```

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

```
def nltk_preprocessing_pipeline(text):
    # Initialize NLTK tools
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    # 1. Text Preprocessing
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE) #remove URLs
    text = re.sub(r'<.*?>', '', text) #remove HTML tags
    text = re.sub(r'@\w+', '', text) #remove mentions
    text = re.sub(r'#\w+', '', text) #remove hashtags
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces

    # 2. Word Tokenization
    tokenized_words = word_tokenize(text)

    # 3. Stopword Removal
    filtered_words = [word for word in tokenized_words if word not in stop_words]

    # 4. Lemmatization
    lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]

    # 5. Rejoin words
    clean_summary = ' '.join(lemmatized_words)

    return clean_summary

print("NLTK preprocessing pipeline function created successfully!")
```

```
NLTK preprocessing pipeline function created successfully!
```

```
df['clean_summaries_pipeline'] = df['summaries'].apply(nltk_preprocessing_pipeline)
print("\nComparison of previous clean_summaries and new clean_summaries_pipeline (first 5 rows):")
print(df[['clean_summaries_pipeline']].head())
```

```

Comparison of previous clean_summaries and new clean_summaries_pipeline (first 5 rows):
                            clean_summaries_pipeline
0  stereo matching one widely used technique infe...
1  recent advancement artificial intelligence ai ...
2  paper proposed novel mutual consistency networ...
3  consistency training proven advanced semisuper...
4  ensure safety automated driving correct percep...
```

```
from sklearn.feature_extraction.text import CountVectorizer
# Vectorize the cleaned summaries
count_vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
doc_term_matrix = count_vectorizer.fit_transform(df['clean_summaries_pipeline'])
```

```
import pandas as pd
# Get feature (word) names
feature_names = count_vectorizer.get_feature_names_out()
# Convert sparse matrix to DataFrame
bow_df = pd.DataFrame(doc_term_matrix.toarray(),columns=feature_names)
# Display BoW matrix for top 10 documents
bow_top_10 = bow_df.head(10)
print(bow_top_10)
```

```
   01  011  014  049  059  060  065  084  089  091  ...  xray  xrays  year  \
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     1
5  0   0   0   0   0   0   0   0   0   0  ...    0     0     0
6  0   0   0   0   0   0   0   0   0   0  ...    0     0     1
7  0   0   0   0   0   0   0   0   0   0  ...    1     0     0
8  0   0   0   0   0   0   0   0   0   0  ...    0     0     0
9  0   0   0   0   0   0   0   0   0   0  ...    0     0     0

   yes  yield  yielded  yielding  youtube  youtubevos  zurich
0    0      0        0         0        0           0       0
1    0      0        0         0        0           0       0
2    0      0        0         0        0           0       0
3    0      0        0         0        0           0       0
4    0      0        0         0        0           0       0
5    0      0        0         0        0           0       0
6    0      0        0         0        0           0       0
7    0      0        0         0        0           0       0
8    0      0        0         0        0           0       0
9    0      0        0         0        0           0       0

[10 rows x 4386 columns]
```

```
from sklearn.decomposition import LatentDirichletAllocation

# Initialize and fit LDA model
num_topics = 2
LDA = LatentDirichletAllocation(n_components=num_topics, random_state=42)
LDA.fit(doc_term_matrix)
```

```
▾             LatentDirichletAllocation                 i ?

LatentDirichletAllocation(n_components=2, random_state=42)
```

```
def display_topics(model, feature_names, num_top_words):
    for topic_idx in range(len(model.components_)):
        print(f"\nTopic {topic_idx}:")

        # Get word weights for this topic
        topic_weights = model.components_[topic_idx]

        # Get indices of words sorted by weight (descending)
        sorted_indices = topic_weights.argsort()[::-1]

        # Take top N words
        top_indices = sorted_indices[:num_top_words]

        # Print top words
        for idx in top_indices:
            print(feature_names[idx], end=" ")
        print()
```

```
# Display top words for each topic
num_top_words = 10
print(f"\nTop {num_top_words} words per topic:")
display_topics(LDA, count_vectorizer.get_feature_names_out(), num_top_words)
```

```

Top 10 words per topic:

Topic 0:
method network feature model proposed approach result algorithm based semantic

Topic 1:
network method model learning data deep training medical task performance
```

```
# Assign topics to each document
document_topics = LDA.transform(doc_term_matrix)
df['topic'] = document_topics.argmax(axis=1)

print("\nDataFrame with assigned topics (first 5 rows):")
print(df[['clean_summaries_pipeline', 'topic']].head())
```

```

DataFrame with assigned topics (first 5 rows):
                         clean_summaries_pipeline  topic
0  stereo matching one widely used technique infe...      1
1  recent advancement artificial intelligence ai ...      1
```

```
2  paper proposed novel mutual consistency networ...      1
3  consistency training proven advanced semisuper...      1
4  ensure safety automated driving correct percep...      1
```

NMF OF KAGGLE DATASET USING BAG OF WORDS

```
from sklearn.decomposition import NMF

# Initialize NMF model
num_topics = 2
nmf_model = NMF(n_components=num_topics, random_state=42)

# Fit the NMF model to the document-term matrix
nmf_model.fit(doc_term_matrix)
print("NMF model initialized and fitted successfully.")
```

```
NMF model initialized and fitted successfully.
```

```
num_top_words = 10
print(f"\nTop {num_top_words} words per topic (NMF):")
display_topics(nmf_model, count_vectorizer.get_feature_names_out(), num_top_words)
```

```

Top 10 words per topic (NMF):

Topic 0:
method model learning data training medical deep domain approach performance

Topic 1:
network neural architecture feature task convolutional deep proposed propose performance
```

```
# Assign topics to each document
document_topics = nmf_model.transform(doc_term_matrix)
df['topic'] = document_topics.argmax(axis=1)

print("\nDataFrame with assigned topics (first 5 rows):")
print(df[['clean_summaries_pipeline', 'topic']].head())
```

```

DataFrame with assigned topics (first 5 rows):
                            clean_summaries_pipeline  topic
0  stereo matching one widely used technique infe...      1
1  recent advancement artificial intelligence ai ...      0
2  paper proposed novel mutual consistency networ...      0
3  consistency training proven advanced semisuper...      0
4  ensure safety automated driving correct percep...      0
```

NMF OF SAMPLE DATA USING TF-IDF

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

# Your small dataset
df = pd.read_excel('/content/drive/MyDrive/LDA-Data.xlsx')
documents = df["News"]

# TF-IDF with no filtering, keep all words
tfidf_vectorizer = TfidfVectorizer(
    stop_words=None,    # include all words, no stop word removal
    min_df=1,           # include words appearing in at least 1 document
    max_df=1.0,         # include words appearing in all documents
    max_features=None,  # no limit on vocab size
    ngram_range=(1, 1)  # unigrams only for simplicity
)

tfidf = tfidf_vectorizer.fit_transform(documents)
vocab = tfidf_vectorizer.get_feature_names_out()

print(f"Vocabulary size: {len(vocab)}")
print("Vocabulary words:", vocab)

# NMF with 2 topics
nmf_model = NMF(n_components=2, random_state=42)
```

```
W = nmf_model.fit_transform(tfidf)
H = nmf_model.components_

# Print topics in descending order of word importance
n_top_words = 5
print("\nTopics discovered by NMF:")
for topic_idx, topic in enumerate(H):
    top_indices = topic.argsort()[-n_top_words:][::-1]
    top_words = [vocab[i] for i in top_indices]
    print(f"Topic {topic_idx + 1}: {top_words}")
```

```
Vocabulary size: 15
Vocabulary words: ['bjp' 'bumra' 'century' 'congress' 'elections' 'form' 'government' 'in'
 'match' 'scored' 'state' 'took' 'virat' 'wicket' 'won']

Topics discovered by NMF:
Topic 1: ['in', 'match', 'virat', 'scored', 'took']
Topic 2: ['state', 'form', 'government', 'congress', 'elections']
```

NMF OF KAGGLE DATASET USING TF-IDF

```
import zipfile
import os
import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

# --- Load dataset ---
zip_path = "/content/drive/MyDrive/arxiv_data.csv.zip"
extract_path = "/content/arxiv_data"

with zipfile.ZipFile(zip_path, "r") as z:
    z.extractall(extract_path)

csv_file = [f for f in os.listdir(extract_path) if f.endswith(".csv")][0]
df = pd.read_csv(os.path.join(extract_path, csv_file))

# --- Text column (CHANGE if needed) ---
TEXT_COL = "summaries"
texts = df[TEXT_COL].dropna().astype(str)

# --- TF-IDF ---
tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=15   # keeps vocabulary small (like your example)
)

X = tfidf.fit_transform(texts)

vocab = tfidf.get_feature_names_out()

# --- Print vocabulary ---
print(f"Vocabulary size: {len(vocab)}")
print("Vocabulary words:", list(vocab))
print()

# --- NMF ---
n_topics = 2

nmf = NMF(
    n_components=n_topics,
    random_state=42,
    init="nndsvda"
)

nmf.fit(X) # Fit the NMF model
H = nmf.components_ # Get the topic-word matrix

# --- Print topics ---
print("Topics discovered by NMF:")

n_top_words = 5
```

```
for i, topic in enumerate(H):
    top_indices = topic.argsort()[-n_top_words:][::-1]
    top_words = [vocab[j] for j in top_indices]
    print(f"Topic {i+1}: {top_words}")
```

```
Vocabulary size: 15
Vocabulary words: ['based', 'data', 'deep', 'image', 'images', 'learning', 'method', 'methods', 'model', 'models', 'network',

Topics discovered by NMF:
Topic 1: ['learning', 'data', 'model', 'models', 'based']
Topic 2: ['image', 'images', 'network', 'method', 'proposed']
```

```
import zipfile
import os
import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

# --- Load dataset ---
zip_path = "/content/drive/MyDrive/arxiv_data.csv.zip"
extract_path = "/content/arxiv_data"

# Extract the zip file
with zipfile.ZipFile(zip_path, "r") as z:
    z.extractall(extract_path)

# Find the CSV file within the extracted folder
csv_file = [f for f in os.listdir(extract_path) if f.endswith(".csv")][0]
df = pd.read_csv(os.path.join(extract_path, csv_file))

# --- Text column ---
TEXT_COL = "summaries"
texts = df[TEXT_COL].dropna().astype(str)

# --- TF-IDF Vectorization ---
tfidf = TfidfVectorizer(
    stop_words="english", # Remove common English stop words
    max_features=15       # Keep only the top 15 most frequent words
)

X = tfidf.fit_transform(texts) # Fit TF-IDF to the text data and transform it

vocab = tfidf.get_feature_names_out() # Get the vocabulary (words) from the TF-IDF model

# --- Print vocabulary ---
print(f"Vocabulary size: {len(vocab)}")
print("Vocabulary words:", list(vocab))
print()

# --- NMF Model ---
n_topics = 2 # Define the number of topics to extract

nmf = NMF(
    n_components=n_topics,
    random_state=42, # Set a random state for reproducibility
    init="nndsvda"   # Initialization method for better convergence
)

nmf.fit(X) # Fit the NMF model to the TF-IDF matrix
H = nmf.components_ # Get the topic-word matrix (each row represents a topic, each column a word)

# --- Print topics ---
print("Topics discovered by NMF:")

n_top_words = 5 # Number of top words to display for each topic

for i, topic in enumerate(H):
    # Get indices of words sorted by their importance in the topic (descending)
    top_indices = topic.argsort()[-n_top_words:][::-1]
    # Get the actual words using the vocabulary
    top_words = [vocab[j] for j in top_indices]
    print(f"Topic {i+1}: {top_words}")
```

```
Vocabulary size: 15
Vocabulary words: ['based', 'data', 'deep', 'image', 'images', 'learning', 'method', 'methods', 'model', 'models', 'network',

Topics discovered by NMF:
Topic 1: ['learning', 'data', 'model', 'models', 'based']
Topic 2: ['image', 'images', 'network', 'method', 'proposed']
```