Text Mining Tutorial 3:

Textual Data Representation

Prof. Hsing-Kuo Pao
Teaching Assistant: Ghaluh Indah Permata Sari

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

- Introduction to Information Retrieval (Basic)
- Traditional Model
  - Vector Space Model (VSM) + TF*IDF
  - Singular Value Decomposition (SVD)
  - Latent Semantic Indexing (LSI)

What is Information Retrieval (IR)?

- **Purpose**: retrieve and ranks the documents that are relevant to the user queries.
- To implement the process, they attribute a value called as **Retrieval Status Value (RSV)** to each candidate document. Afterwards, they rank documents with respect to document retrieval status values (RSV).

IR applications

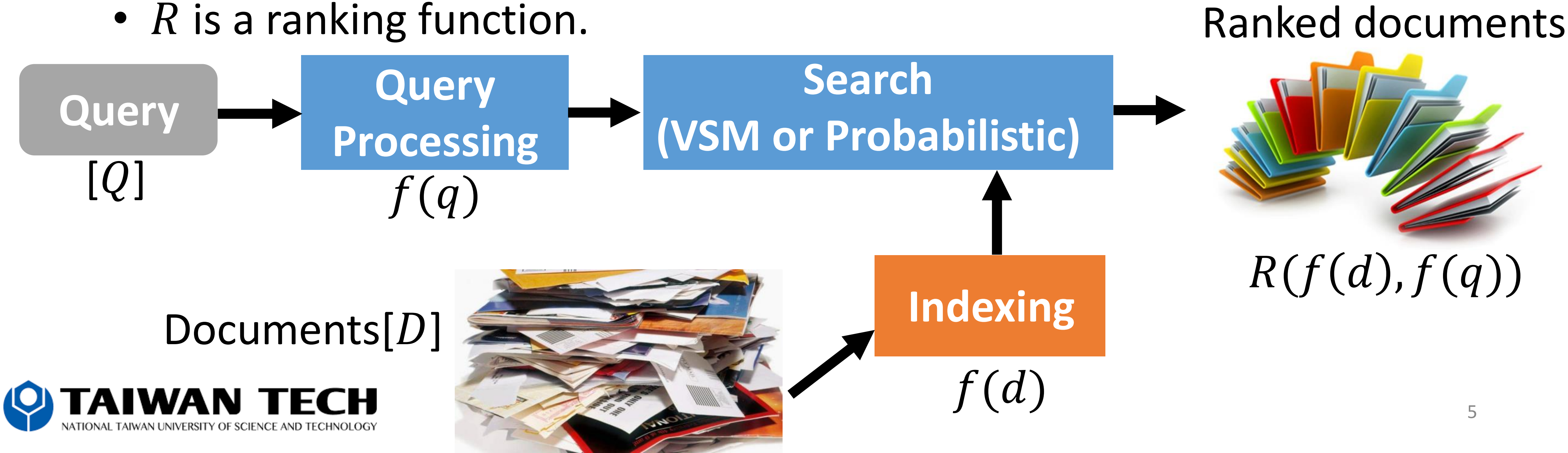
Search engine

Recommender System

IR Pipeline

Formal Expression $[D, Q, f, R]$

- D is a set of documents in the collection $D = \{d_1, \cdots, d_n\}$.
- Q is a set of queries $Q = \{q_1, \cdots, q_m\}$.
- f is a function that translates the D and Q into a representations.
- R is a ranking function.



Query

$[Q]$

Query Processing

$f(q)$

Search (VSM or Probabilistic)

Ranked documents

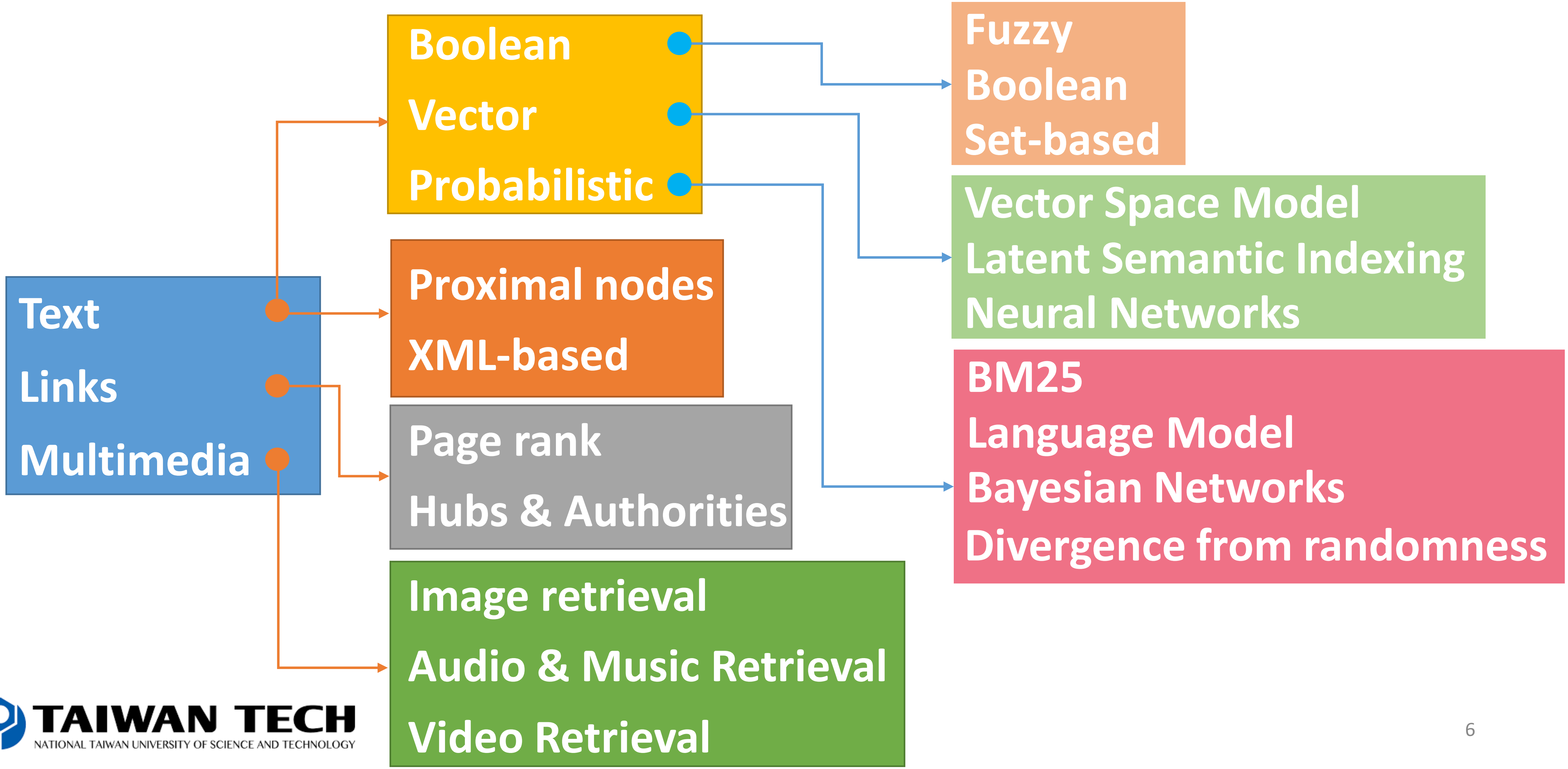
$R(f(d), f(q))$

Documents$[D]$

Indexing

$f(d)$

IR Taxonomy



Text
Links
Multimedia

Boolean
Vector
Probabilistic

Proximal nodes
XML-based

Page rank
Hubs & Authorities

Image retrieval
Audio & Music Retrieval
Video Retrieval

Fuzzy
Boolean
Set-based

Vector Space Model
Latent Semantic Indexing
Neural Networks

BM25
Language Model
Bayesian Networks
Divergence from randomness

Reminder

Bernoulli Distribution for index term

	Relevant	Non-relevant	All documents
Documents that contain w_i	r_i	$n_i - r_i$	n_i
Documents that do not contain w_i	$R_q - r_i$	$N - n_i - (R_q - r_i)$	$N - n_i$
All documents	R_q	$N - R_q$	N

For a given query, if we have:

- N be the number of documents in the collection
- n_i be the number of documents that contain term w_i
- R_q be the total number of relevant documents to query q
- r_i be the number of relevant documents that contain term w_i

Index term

- Each document is represented by a set of representative keywords or index terms
  - An index term is a word or group of consecutive words in a document
- A pre-selected set of index terms can be used to summarize the document's contents
  - Lexicon (the complete set of words)
  - Vocabulary (the subset of words or lexical items used by a particular individual)
- However, it might be interesting to assume that all words are index terms (full text representation)

Index term: Boolean model (SOTA)

- It is a simple retrieval model based on set theory and boolean algebra.
- The retrieval strategy is based on binary decision criteria.
- The boolean model considers that index terms are present (1) or absent (0) in a document.

Boolean model: case example

d1 = ‘ the way to avoid linearly scanning is to index the documents in advance‘

d2 = ‘ the model views each document as just a set of words‘

d3 = ‘ we will discuss and model these size assumption‘

Boolean value

Vocabulary/ Lexicon

Terms (t_i)	d_1	d_2	d_3
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: case example

Given a query: “ way “

➡ $\text{way} = [1\ \ 0\ \ 0]$

$$\begin{array}{ccc} \vdots & \vdots & \vdots \\ d_1 & d_2 & d_3 \end{array}$$

➡ ∴ answer = d_1

Boolean value

Terms (t_i)	d_1	d_2	d_3
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: case example “logical operators”

Given a query: non “ way “

➡ $\neg way$ = ¬[1 0 0] = [0 1 1]

➡ ∴ answer = d_2 & d_3

Boolean value

Terms (t_i)	d_1	d_2	d_3
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: case example “logical operators”

Given a query: “document” and “model“

➡ $document \ \wedge model =$

$[1\ 1\ 0] \wedge [0\ 1\ 1] = [0\ 1\ 0]$

➡ ∴ answer = d_2

Boolean value

Terms (t_i)	d_1	d_2	d_3
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: case example “logical operators”

Given a query: “avoid” or “view“

➡ $avoid \lor view =$
[1 0 0] ∨ [0 1 0] = [1 1 0]

➡ ∴ answer = d_1 & d_2

Boolean value

Terms (t_i)	$\boldsymbol{d_1}$	$\boldsymbol{d_2}$	$\boldsymbol{d_3}$
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: case example “logical operators”

Given a query:
“avoid” and (“view“ or non “model”)

➡ $avoid \ \wedge (view \vee \neg model) =$
[1 0 0] ∧ ([0 1 0] ∨ [1 0 0]) =
[1 0 0] ∧ [1 1 0]
[1 0 0]

➡ ∴ answer = d_1

Boolean value

Terms (t_i)	d_1	d_2	d_3
⋮			
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0
⋮			

Boolean model: drawbacks

- Retrieval based on binary decision criteria with no notion of partial matching.
- No ranking/ grading scale for the documents is provided.
- The information need must be translated into a Boolean expression, which most users are unhandy.
- The model frequently returns either too few or too many documents in response to a user query.

Quick test

Given a query:
“discuss” or “advance” or (“view“ and non “model”)

What could be the result?

Terms (t_i)	d_1	d_2	d_3
way	1	0	0
document	1	1	0
model	0	1	1
avoid	1	0	0
view	0	1	0
discuss	0	0	1
advance	1	0	0

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Index term: Term frequency

- **Term frequency:** Term frequency tells you how much a term occurs in a document.

TF*IDF = TF(t,d) · IDF(t)

(Term Frequency/ tf)

$$tf = \frac{count\ of\ term\ (t)\ in\ doc.}{num.of\ words\ in\ doc.}$$

(Inverse Document Frequency/ idf)

$$idf = log\left(\frac{docs.in\ corpus}{num.of\ docs.where\ t\ appears}\right)$$

Source paper: Using TF-IDF to Determine Word Relevance in Document Queries (2003)

Term frequency: the tf variant

Weighting scheme	**tf weighting**
Binary	0, 1
Raw count	$tf_{t,d}$
Term frequency	$\frac{tf_{t,d}}{\sum_{T\in d} tf_{T,d}}$
Log normalization	$log\left(1+tf_{t,d}\right)$
Double normalization To observe higher term frequencies in longer documents (higher term keep repeated)	$0.5+0.5\frac{0.5+\frac{0.5\times tf_{t,d}}{max_{tf}}}{0.5+0.5\times\frac{len_d}{avg_{doclen}}}$
Double normalization K Smoothing value	$K+(1-K)\frac{tf_{t,d}}{tf_{max}(d)'}$

T= total number of terms in doc.

max_{tf}=max. term freq. in doc.

len_d= total number of terms in doc.

avg_{doclen}= avg. doc. length in corpus

$K = 0\sim1, best\ 0.4$

$tf_{max}(d)'$= most frequently occurring term

Term frequency: the idf variant

Weighting scheme	**idf weighting**
unary	1
Inverse document frequency	$log\frac{N}{n_t} = -log\frac{n_t}{N}$
Inverse document frequency smooth To avoid non-zero IDF	$log\left(\frac{N+1}{n_t+1}\right)+1$
Inverse document frequency max To mitigate the impact of outlier doc. frequencies on IDF value	$log\left(\frac{max_{(t'\epsilon d)}n_{t'}}{1+n_t}\right)$
Probabilistic inverse document frequency To mitigate infrequent doc. in corpus but frequent in individual doc.	$log\left(\frac{N-n_t+0.5}{n_t+0.5}\right)$

TF*IDF Calculation

Doc 1: It is going to rain today.

Doc 2: Today I am not going outside.

Doc 3: I am going to watch the season premiere.

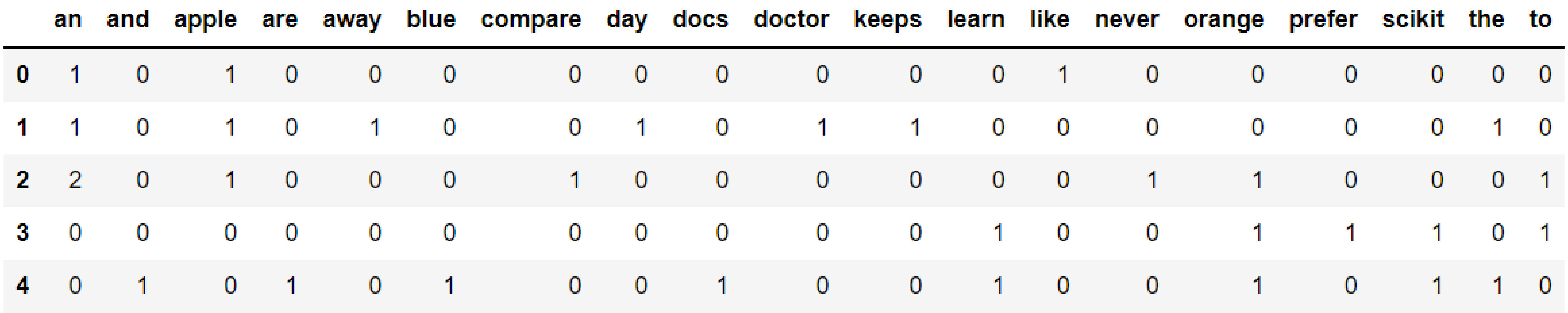
i	Term (t)	Occurrence t_i			Occurrence (t_i)	tf_{t_i}			idf	$tf_{t_i} * idf$		
		d_1	d_2	d_3		d_1	d_2	d_3		d_1	d_2	d_3
1	going	1	1	1	3	0.166667	0.166667	0.125	0	0	0	0
2	to	1	0	1	2	0.166667	0	0.125	0.176091	0.029349	0	0.022011
3	today	1	1	0	2	0.166667	0.166667	0	0.176091	0.029349	0.029349	0
4	i	0	1	1	2	0	0.166667	0.125	0.176091	0	0.029349	0.022011
5	am	0	1	1	2	0	0.166667	0.125	0.176091	0	0.029349	0.022011
6	it	1	0	0	1	0.166667	0	0	0.477121	0.07952	0	0
7	is	1	0	0	1	0.166667	0	0	0.477121	0.07952	0	0
8	rain	1	0	0	1	0.166667	0	0	0.477121	0.07952	0	0

Term frequency: code example

```
# Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Define the documents
corpus = ["I'd like an apple",
            "An apple a day keeps the doctor away",
            "Never compare an apple to an orange",
            "I prefer scikit-learn to Orange",
            "The scikit-learn docs are Orange and Blue"]

# Create the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(corpus) #(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want to see the word frequencies.
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names())

df
```

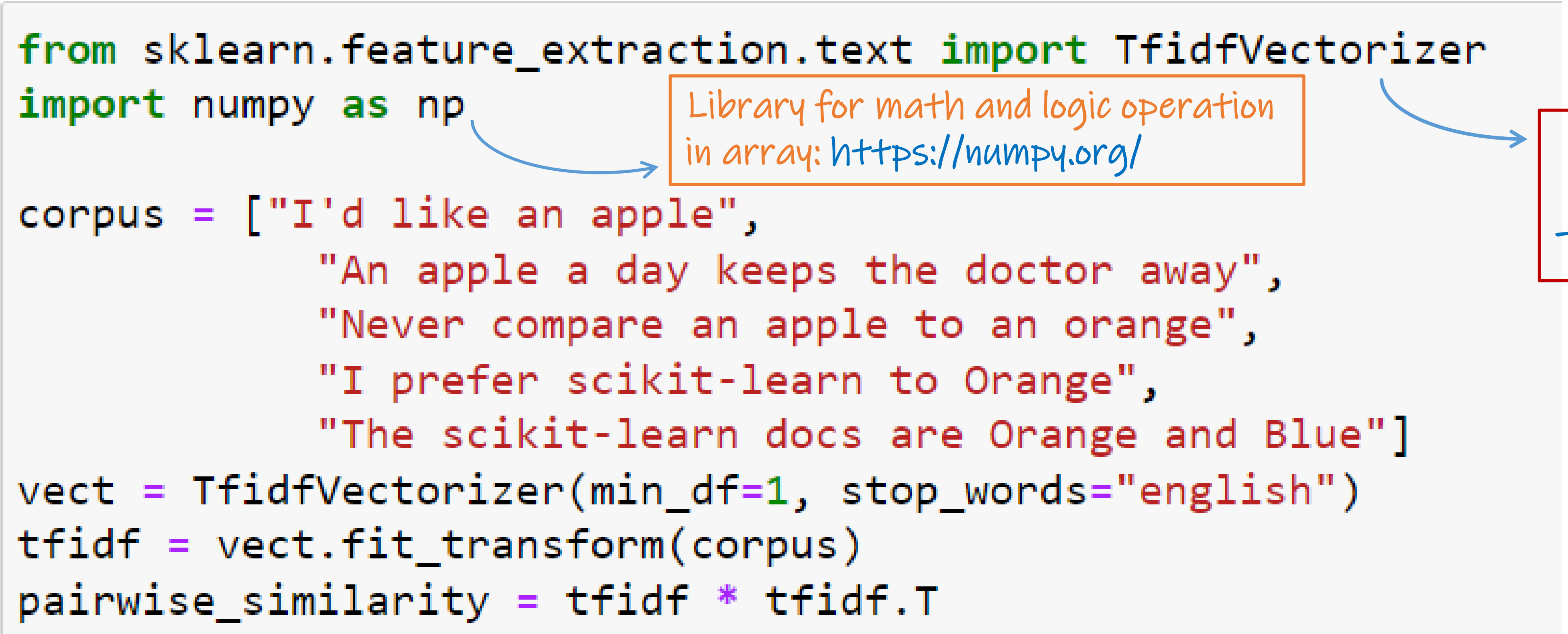
Term frequency: code example

Output



	an	and	apple	are	away	blue	compare	day	docs	doctor	keeps	learn	like	never	orange	prefer	scikit	the	to
0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	1	0
2	2	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	1
4	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0

Tf*Idf: code example



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

corpus = ["I'd like an apple",
            "An apple a day keeps the doctor away",
            "Never compare an apple to an orange",
            "I prefer scikit-learn to Orange",
            "The scikit-learn docs are Orange and Blue"]
vect = TfidfVectorizer(min_df=1, stop_words="english")
tfidf = vect.fit_transform(corpus)
pairwise_similarity = tfidf * tfidf.T
```

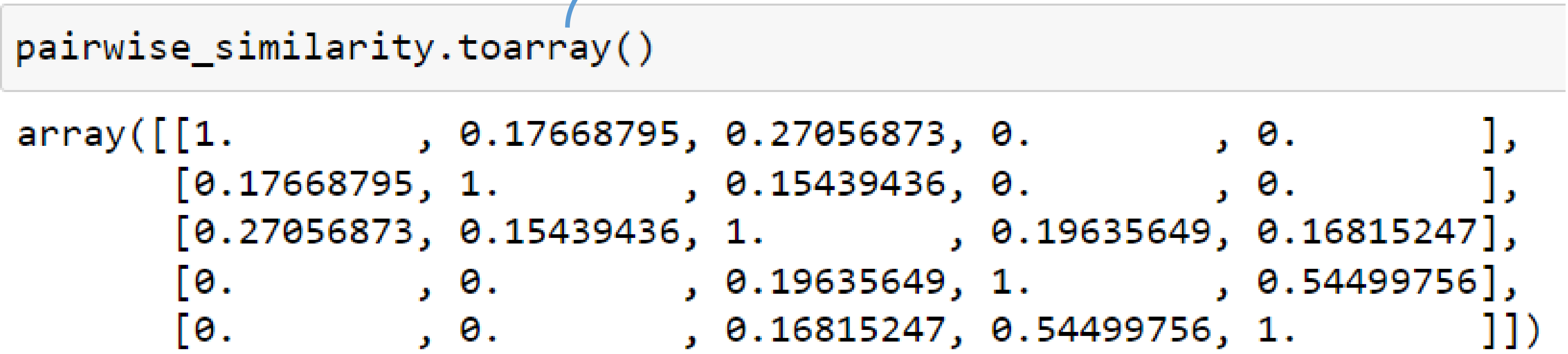
Library for math and logic operation in array: https://numpy.org/

CountVectorizer
+
TfidfTransformer

Sparse matrix ➔ square in shape;
number of rows & columns == to the number of documents in the corpus

Tf*Idf: code example

convert the sparse
matrix to an array

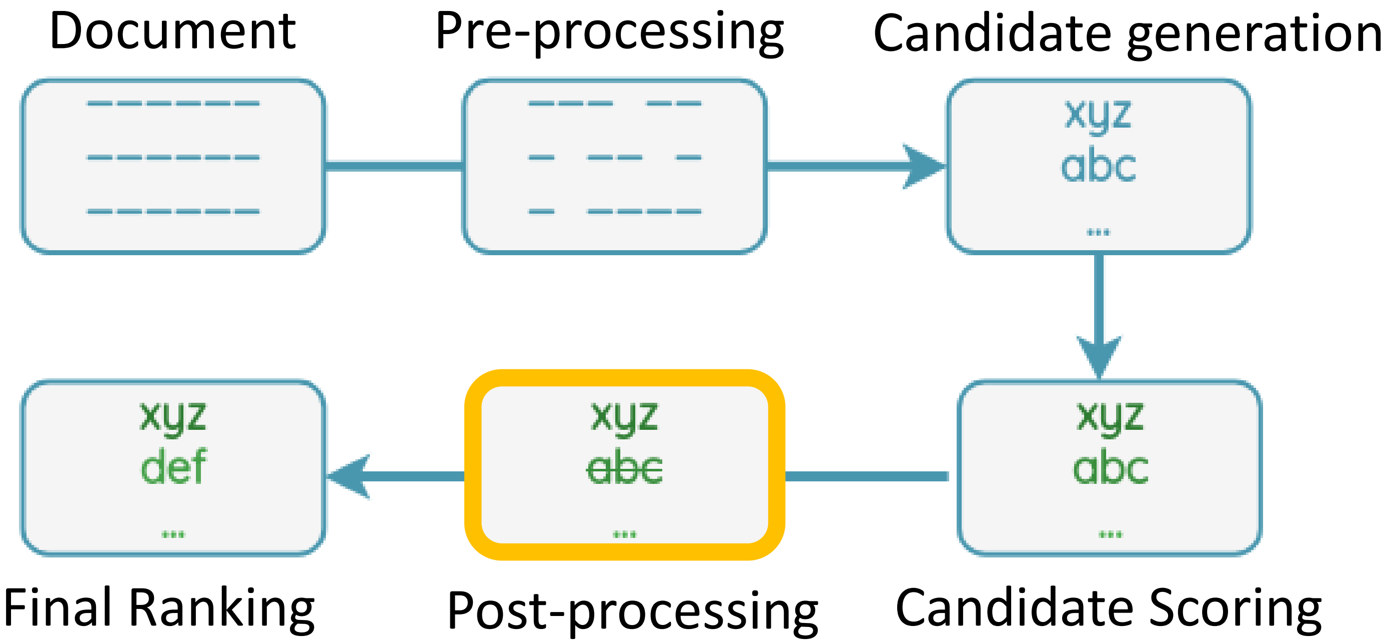


```
pairwise_similarity.toarray()
```

```
array([[1.        , 0.17668795, 0.27056873, 0.        , 0.        ],
       [0.17668795, 1.        , 0.15439436, 0.        , 0.        ],
       [0.27056873, 0.15439436, 1.        , 0.19635649, 0.16815247],
       [0.        , 0.        , 0.19635649, 1.        , 0.54499756],
       [0.        , 0.        , 0.16815247, 0.54499756, 1.        ]])
```

Statistical Approach: Bag-of-Words (BoW)

Keywords Extraction Pipeline



Bag-of-Words (BoW): Example

1.

OLD

$$TF*IDF = TF(t,d) \cdot IDF(t)$$

(Term Frequency)

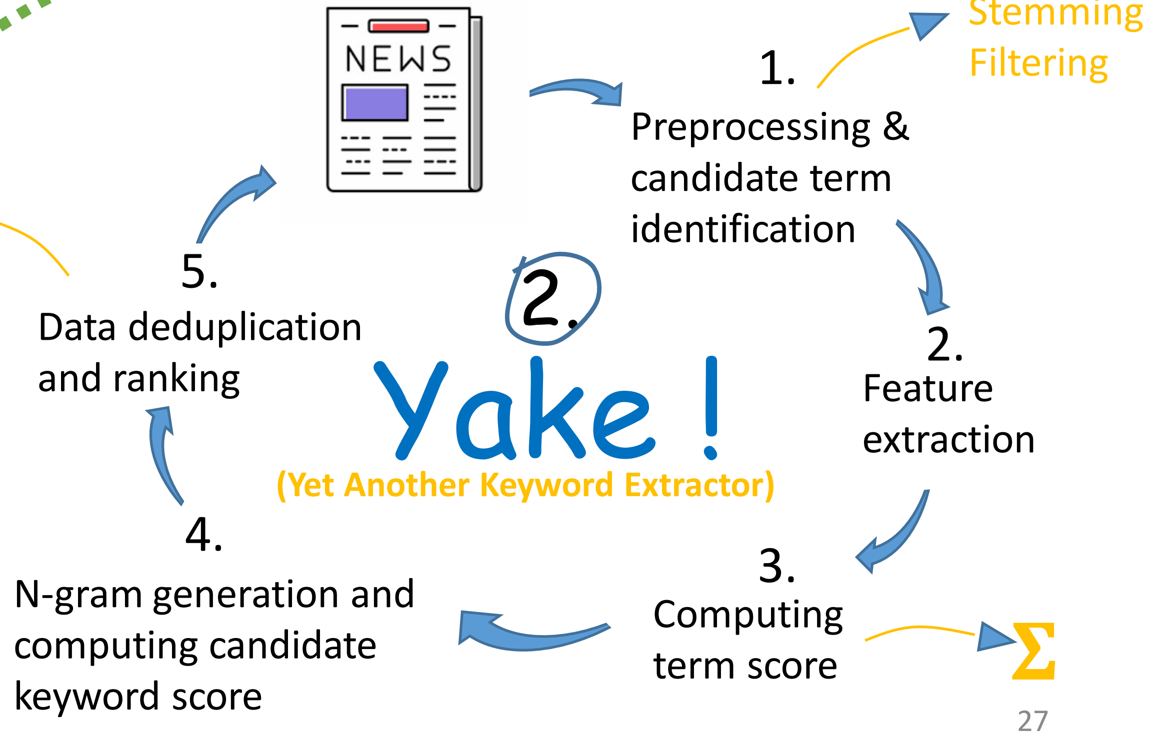
$$tf = \frac{count\ of\ t\ in\ doc.}{num.\ of\ words\ in\ doc.}$$

(Inverse Document Frequency)

$$idf = log\left(\frac{docs.\ in\ corpus}{num.\ of\ docs.\ where\ t\ appears}\right)$$

Source paper: Using TF-IDF to Determine Word Relevance in Document Queries (2003)

2.



Yake !

(Yet Another Keyword Extractor)

1. Preprocessing & candidate term identification
   - Tokenize
   - Stemming
   - Filtering
2. Feature extraction
3. Computing term score → Σ
4. N-gram generation and computing candidate keyword score
5. Data deduplication and ranking → Select high relevancy

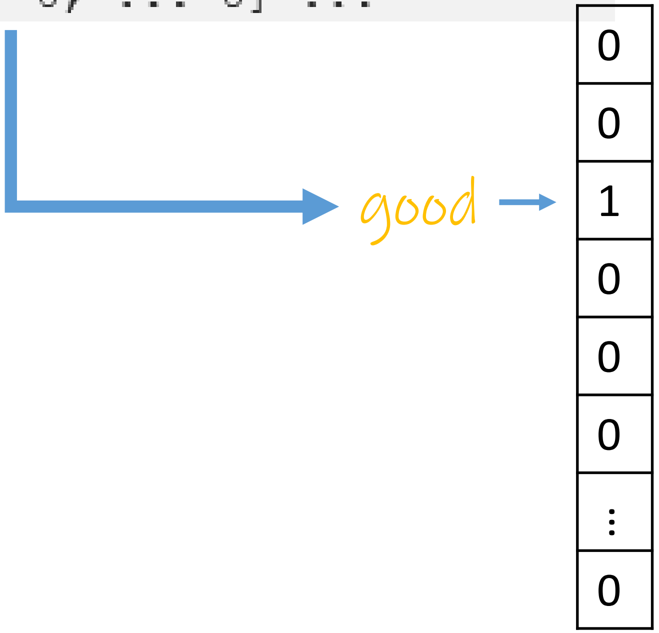
Source paper: YAKE! Keyword extraction from single documents using multiple local features (2019)

Vector Space Model (word embedding)

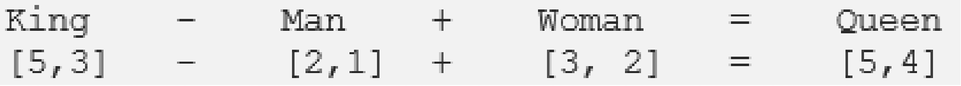
- Word embedding is a technique where individual words are transformed into a numerical representation of the word (a vector).

```
For example: have = [1, 0, 0, 0, 0, 0, ... 0]
a    = [0, 1, 0, 0, 0, 0, ... 0]
good = [0, 0, 1, 0, 0, 0, ... 0]
day  = [0, 0, 0, 1, 0, 0, ... 0] ...
```

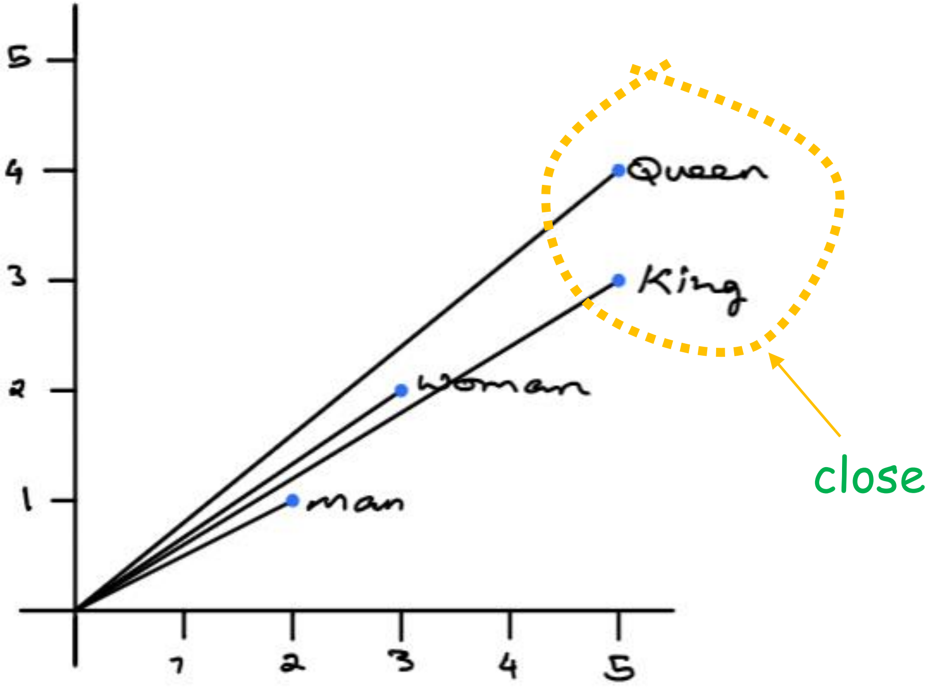
One-hot vector



10,000 positions



King	-	Man	+	Woman	=	Queen
[5,3]	-	[2,1]	+	[3, 2]	=	[5,4]



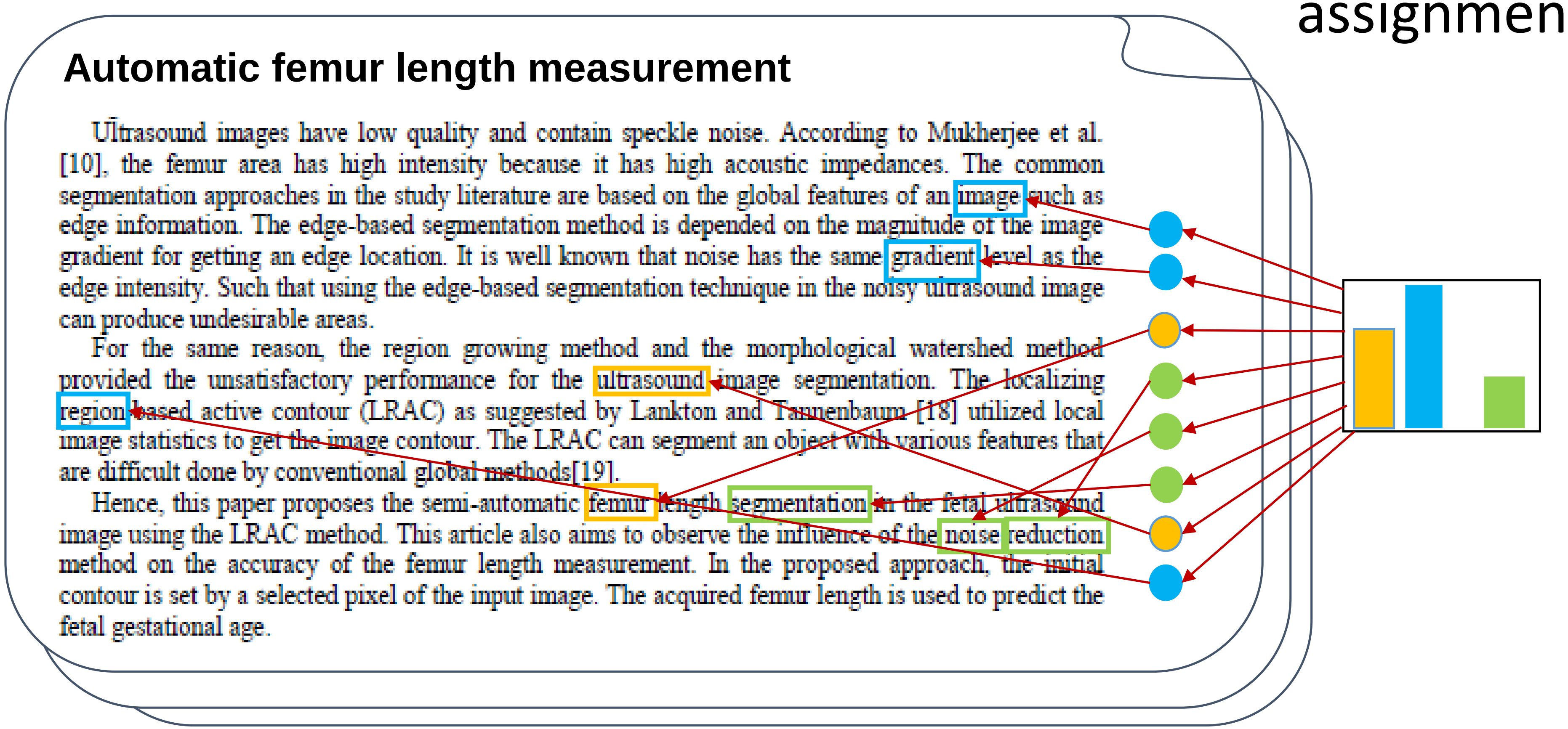
Source: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

Introduction to Topic Modeling

Documents

Topic assignments

Topics



Automatic femur length measurement

Ultrasound images have low quality and contain speckle noise. According to Mukherjee et al. [10], the femur area has high intensity because it has high acoustic impedances. The common segmentation approaches in the study literature are based on the global features of an image such as edge information. The edge-based segmentation method is depended on the magnitude of the image gradient for getting an edge location. It is well known that noise has the same gradient level as the edge intensity. Such that using the edge-based segmentation technique in the noisy ultrasound image can produce undesirable areas.

For the same reason, the region growing method and the morphological watershed method provided the unsatisfactory performance for the ultrasound image segmentation. The localizing region-based active contour (LRAC) as suggested by Lankton and Tannenbaum [18] utilized local image statistics to get the image contour. The LRAC can segment an object with various features that are difficult done by conventional global methods[19].

Hence, this paper proposes the semi-automatic femur length segmentation in the fetal ultrasound image using the LRAC method. This article also aims to observe the influence of the noise reduction method on the accuracy of the femur length measurement. In the proposed approach, the initial contour is set by a selected pixel of the input image. The acquired femur length is used to predict the fetal gestational age.

words	Score
femur	0.04
ultrasound	0.02
genetic	0.01
…	

words	Score
noise	0.02
segmentation	0.01
reduction	0.01
…	

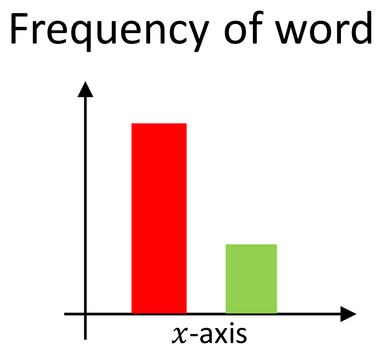
words	Score
region	0.04
gradient	0.02
image	0.01
…	

Introduction to Topic Modeling

Topic Analysis

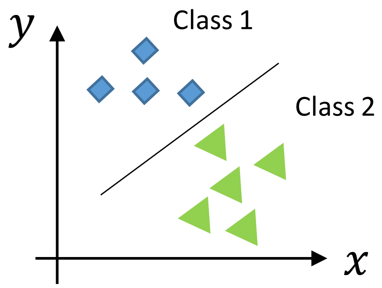
Topic Modeling

- Latent Semantic Analysis (LSA/ LSI)
- Latent Dirichlet Allocation (LDA)
- Non-Negative Matrix Factorization (NNMF)



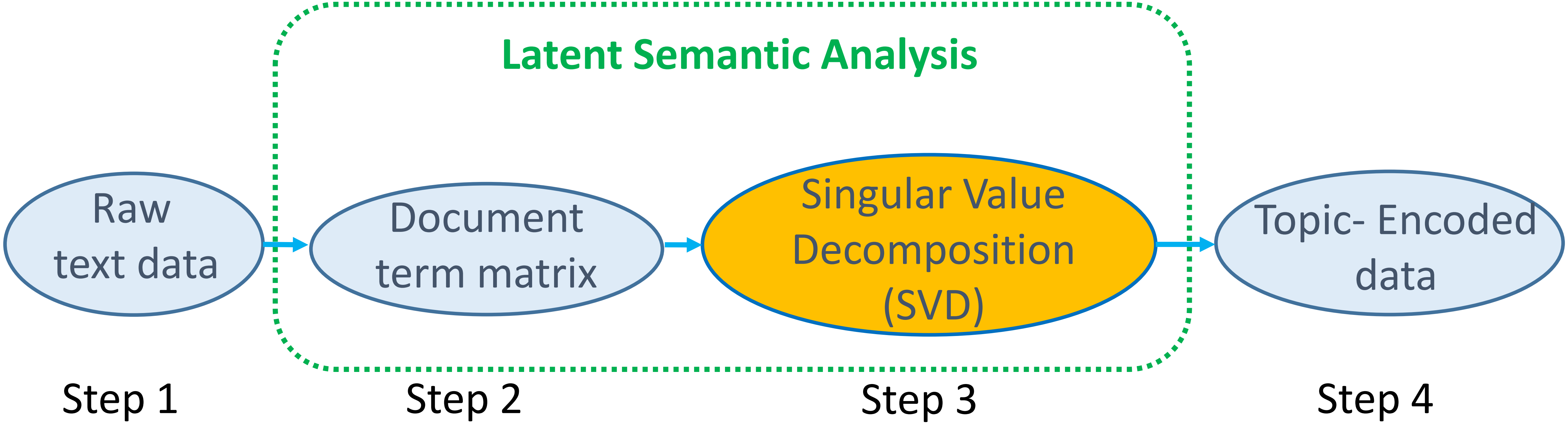
Topic Classification

- Rule-Based system
- Machine Learning
- Deep Learning



Latent Semantic Analysis (LSA)

By Deerwester, Dumais, et al., 1990



Latent Semantic Analysis

Raw text data

Document term matrix

Singular Value Decomposition (SVD)

Topic- Encoded data

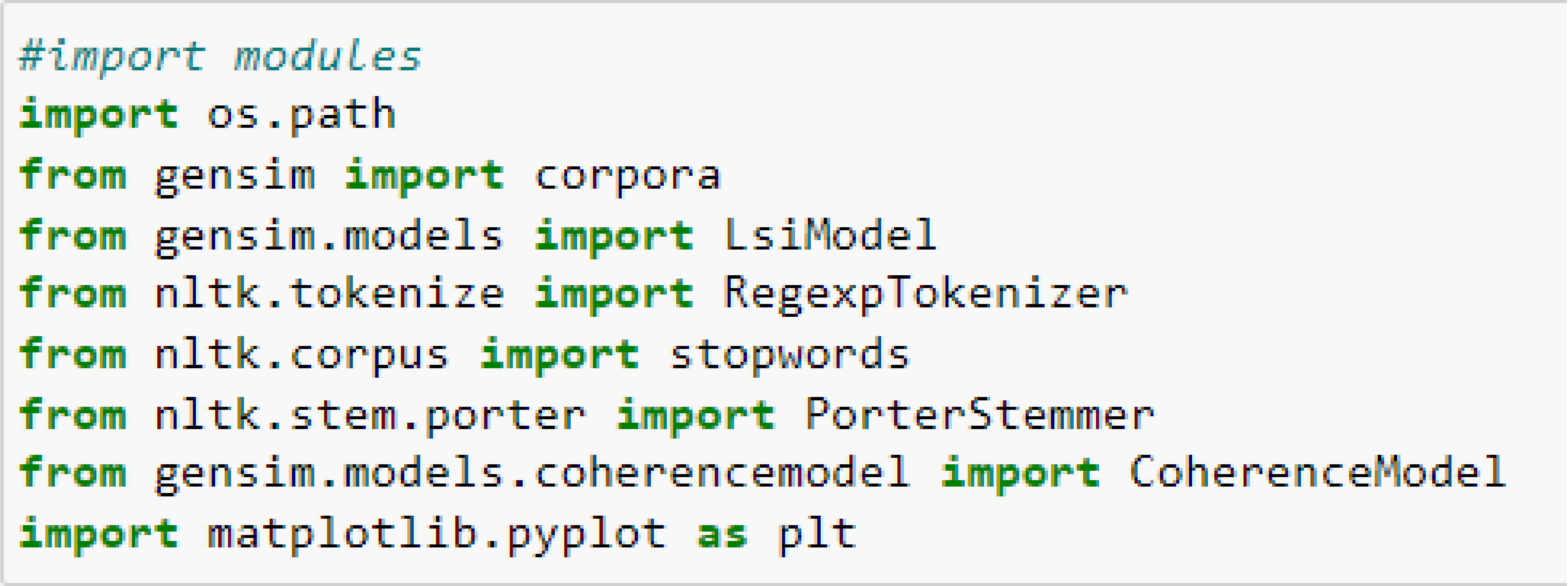
Step 1

Step 2

Step 3

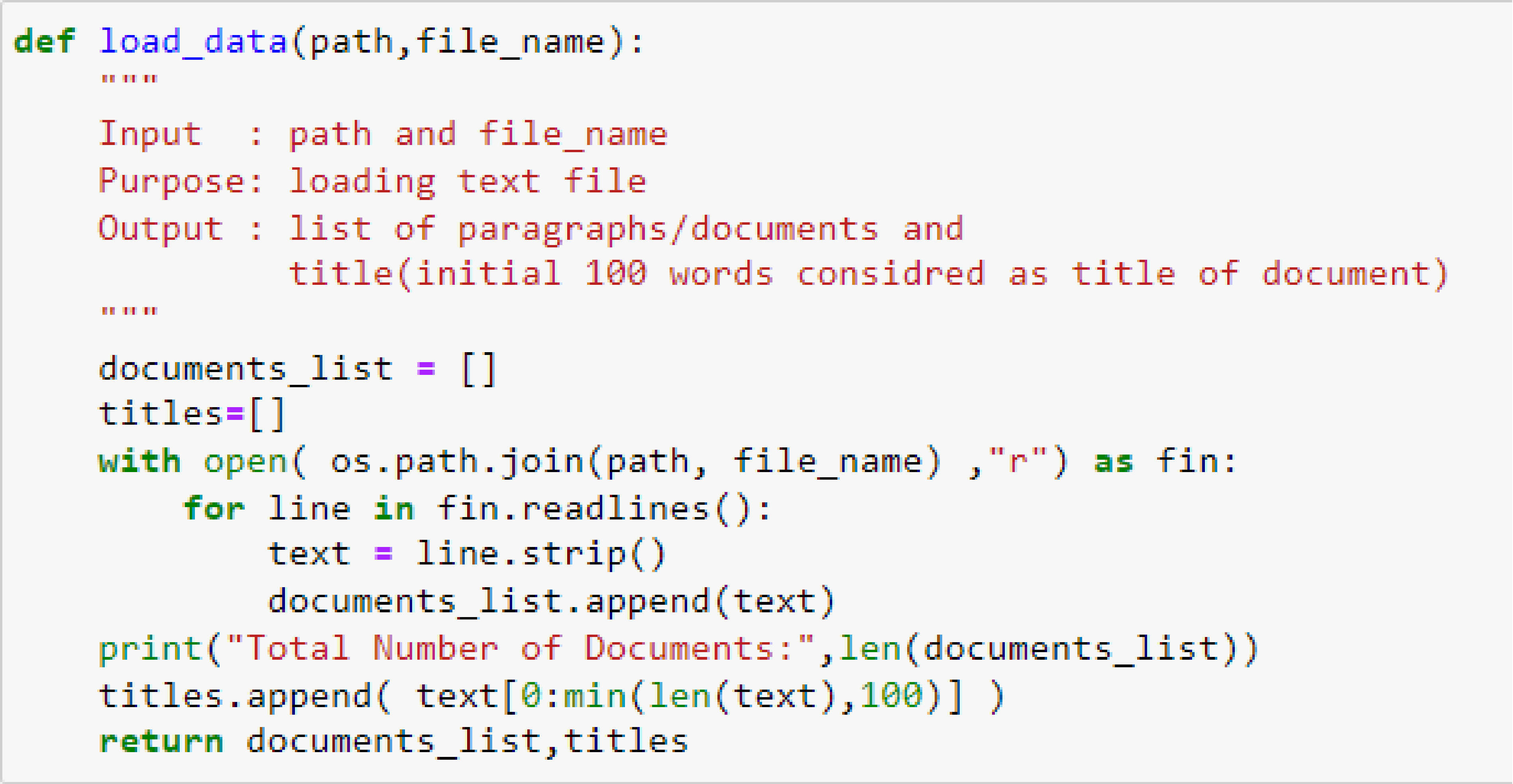
Step 4

Latent Semantic Analysis (LSA): code example



```
#import modules
import os.path
from gensim import corpora
from gensim.models import LsiModel
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
```

Latent Semantic Analysis (LSA): code example

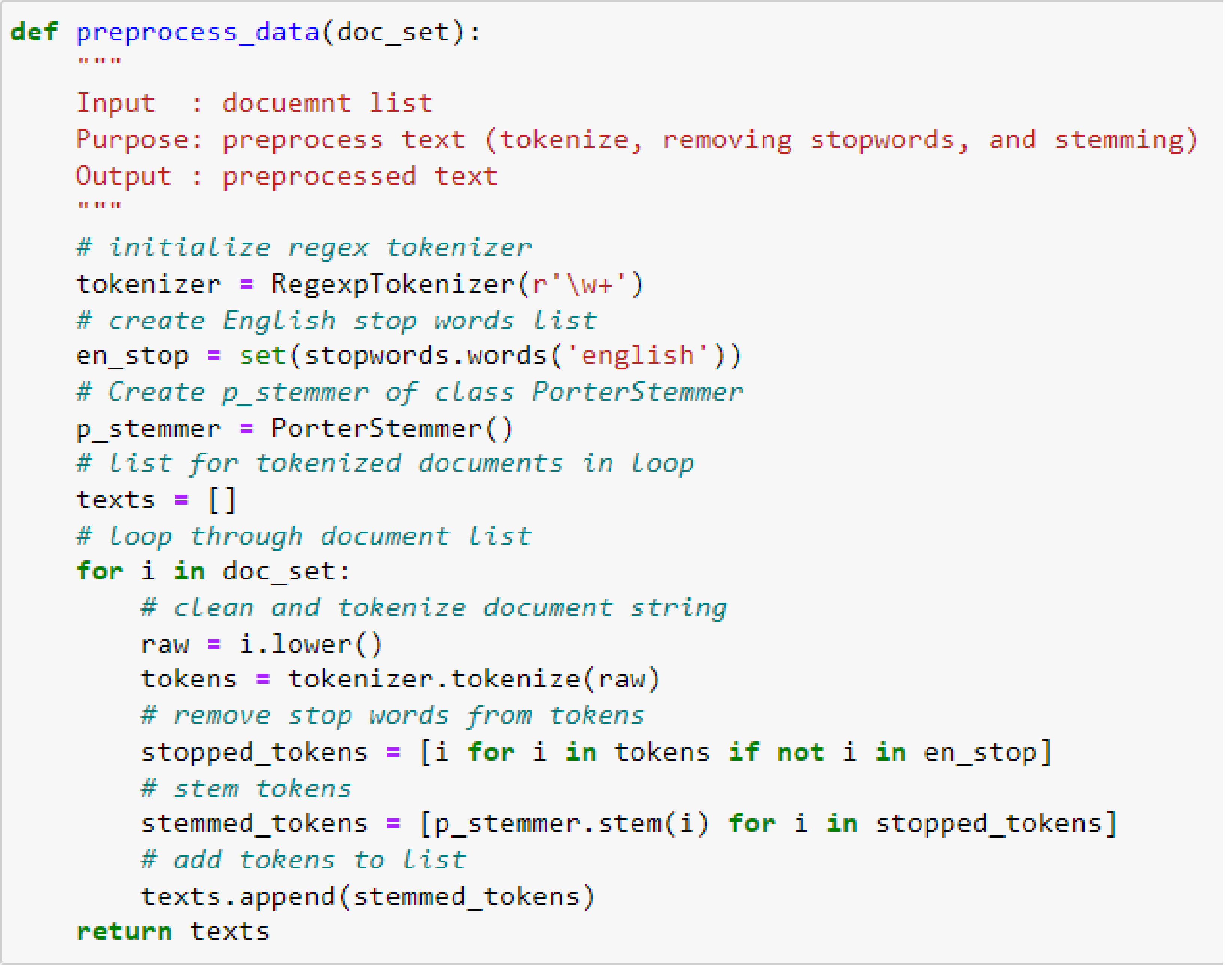


```
def load_data(path,file_name):
    """
    Input  : path and file_name
    Purpose: loading text file
    Output : list of paragraphs/documents and
             title(initial 100 words considred as title of document)
    """
    documents_list = []
    titles=[]
    with open( os.path.join(path, file_name) ,"r") as fin:
        for line in fin.readlines():
            text = line.strip()
            documents_list.append(text)
    print("Total Number of Documents:",len(documents_list))
    titles.append( text[0:min(len(text),100)] )
    return documents_list,titles
```

LSA: code example

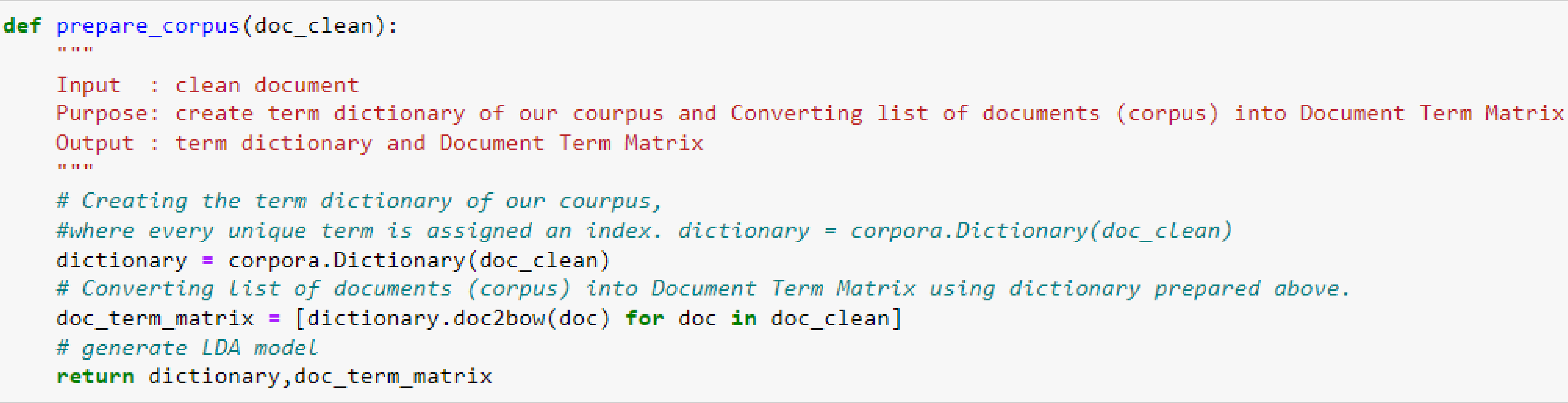
Pre-processing:

- Tokenization
- Stopwords removal
- Lower case



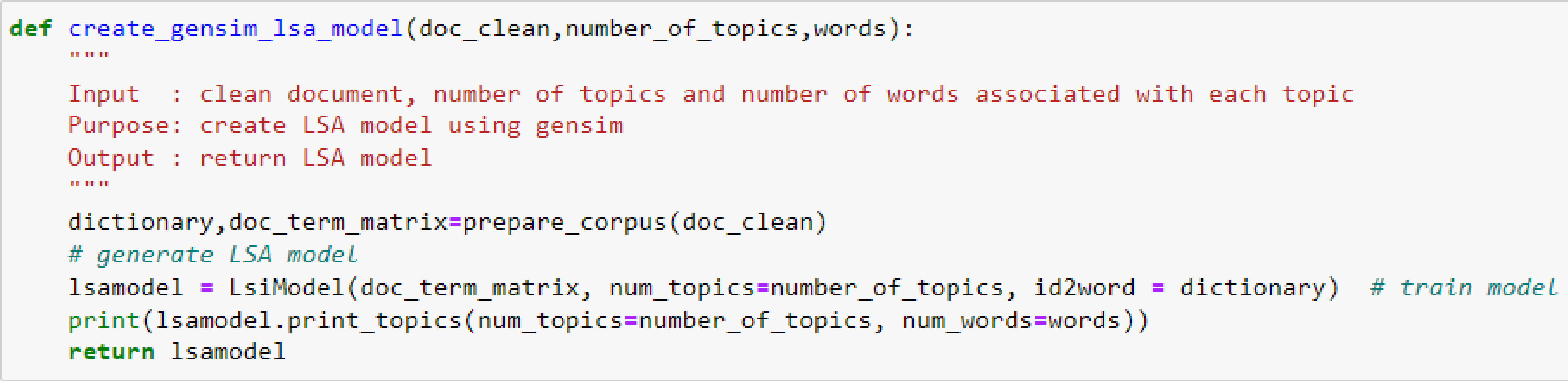
```
def preprocess_data(doc_set):
    """
    Input  : docuemnt list
    Purpose: preprocess text (tokenize, removing stopwords, and stemming)
    Output : preprocessed text
    """
    # initialize regex tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    # create English stop words list
    en_stop = 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
```

Latent Semantic Analysis (LSA): code example



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

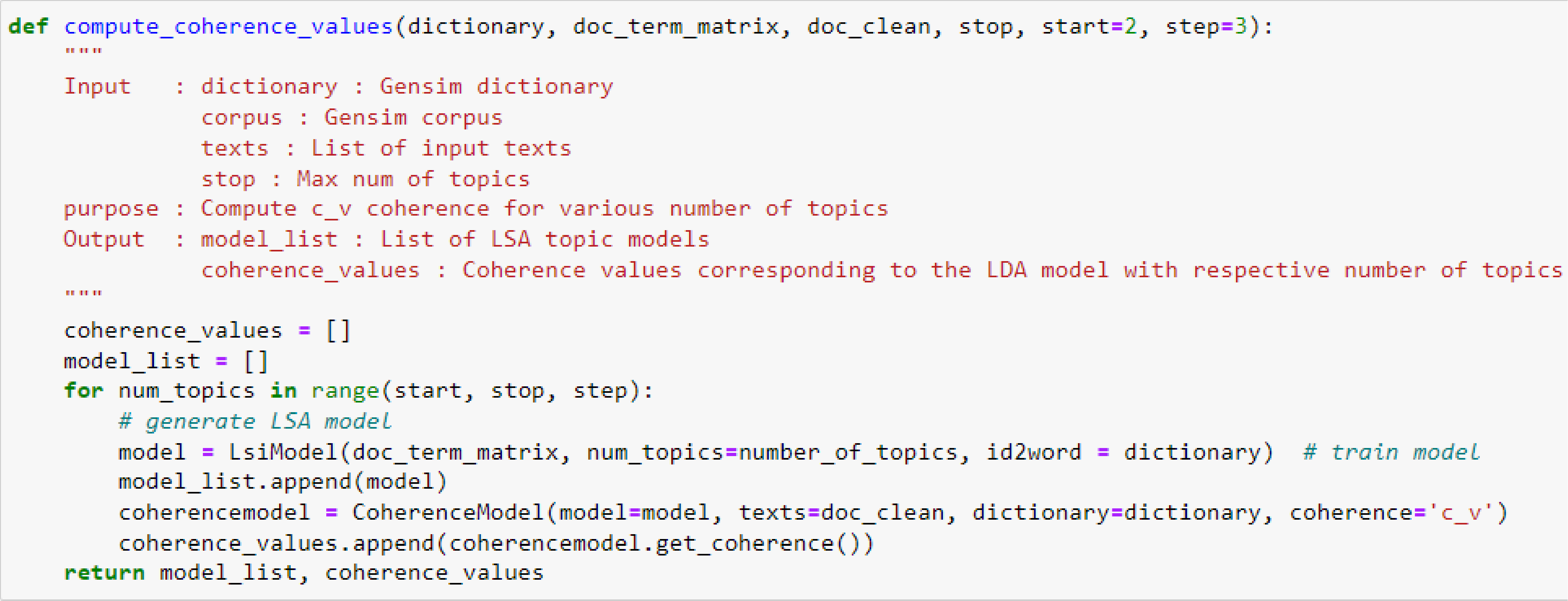
Latent Semantic Analysis (LSA): code example



```
def create_gensim_lsa_model(doc_clean,number_of_topics,words):
    """
    Input  : clean document, number of topics and number of words associated with each topic
    Purpose: create LSA model using gensim
    Output : return LSA model
    """
    dictionary,doc_term_matrix=prepare_corpus(doc_clean)
    # generate LSA model
    lsamodel = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word = dictionary)  # train model
    print(lsamodel.print_topics(num_topics=number_of_topics, num_words=words))
    return lsamodel
```

Source: https://radimrehurek.com/gensim/models/lsimodel.html

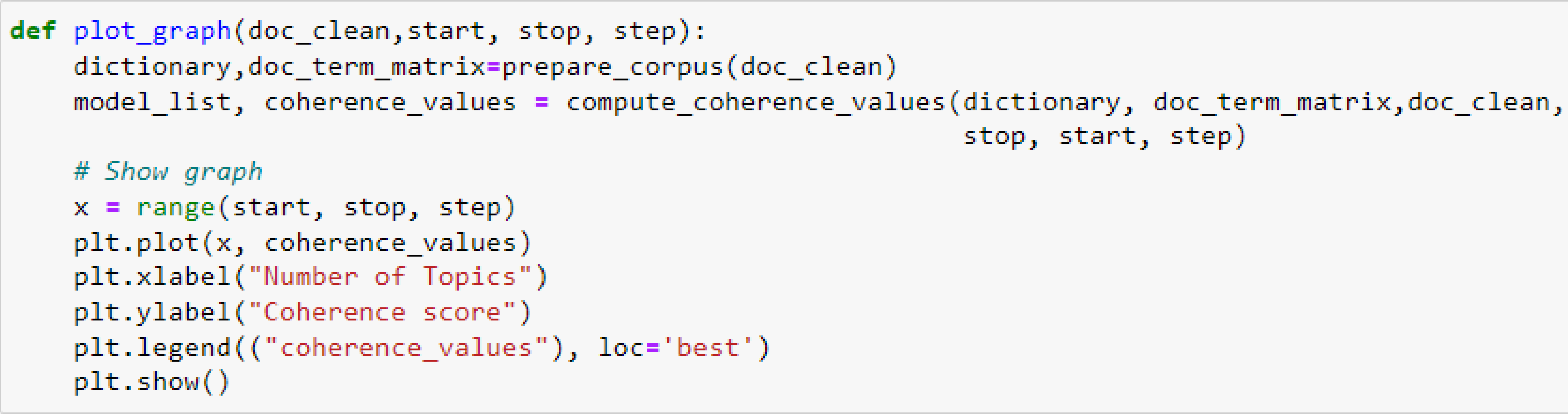
Latent Semantic Analysis (LSA): code example



```
def compute_coherence_values(dictionary, doc_term_matrix, doc_clean, stop, start=2, step=3):
    """
    Input   : dictionary : Gensim dictionary
              corpus : Gensim corpus
              texts : List of input texts
              stop : Max num of topics
    purpose : Compute c_v coherence for various number of topics
    Output  : model_list : List of LSA topic models
              coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, stop, step):
        # generate LSA model
        model = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word = dictionary)  # train model
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=doc_clean, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())
    return model_list, coherence_values
```

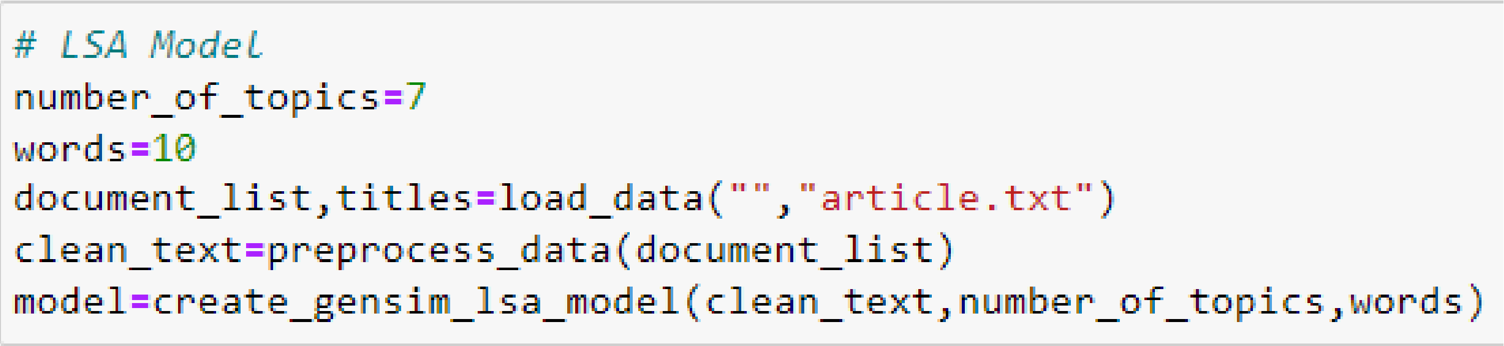
Source: https://radimrehurek.com/gensim/models/coherencemodel.html

Latent Semantic Analysis (LSA): code example

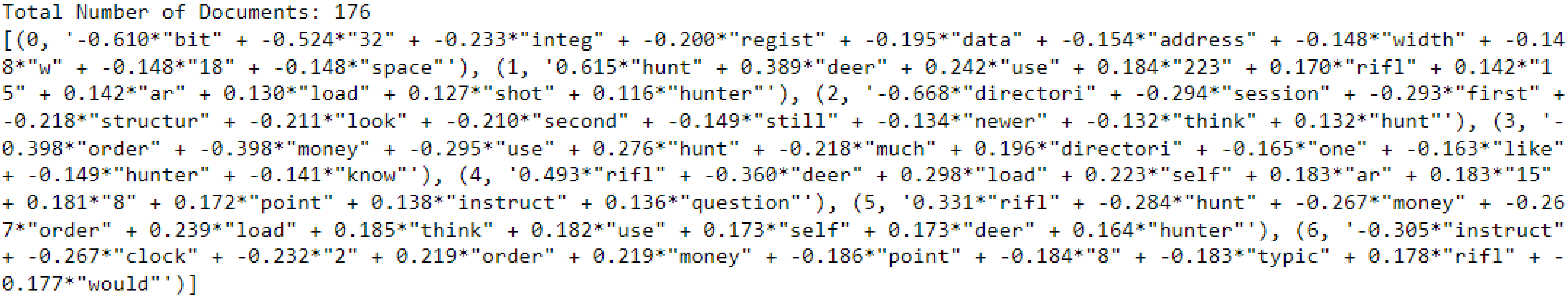


```
def plot_graph(doc_clean,start, stop, step):
    dictionary,doc_term_matrix=prepare_corpus(doc_clean)
    model_list, coherence_values = compute_coherence_values(dictionary, doc_term_matrix,doc_clean,
                                                            stop, start, step)
    # Show graph
    x = range(start, stop, step)
    plt.plot(x, coherence_values)
    plt.xlabel("Number of Topics")
    plt.ylabel("Coherence score")
    plt.legend(("coherence_values"), loc='best')
    plt.show()
```

Latent Semantic Analysis (LSA): code example

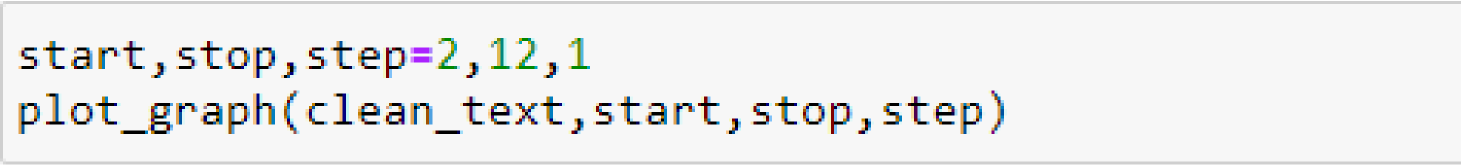


```
# LSA Model
number_of_topics=7
words=10
document_list,titles=load_data("","article.txt")
clean_text=preprocess_data(document_list)
model=create_gensim_lsa_model(clean_text,number_of_topics,words)
```



```
Total Number of Documents: 176
[(0, '-0.610*"bit" + -0.524*"32" + -0.233*"integ" + -0.200*"regist" + -0.195*"data" + -0.154*"address" + -0.148*"width" + -0.14
8*"w" + -0.148*"18" + -0.148*"space"'), (1, '0.615*"hunt" + 0.389*"deer" + 0.242*"use" + 0.184*"223" + 0.170*"rifl" + 0.142*"1
5" + 0.142*"ar" + 0.130*"load" + 0.127*"shot" + 0.116*"hunter"'), (2, '-0.668*"directori" + -0.294*"session" + -0.293*"first" +
-0.218*"structur" + -0.211*"look" + -0.210*"second" + -0.149*"still" + -0.134*"newer" + -0.132*"think" + 0.132*"hunt"'), (3, '-
0.398*"order" + -0.398*"money" + -0.295*"use" + 0.276*"hunt" + -0.218*"much" + 0.196*"directori" + -0.165*"one" + -0.163*"like"
+ -0.149*"hunter" + -0.141*"know"'), (4, '0.493*"rifl" + -0.360*"deer" + 0.298*"load" + 0.223*"self" + 0.183*"ar" + 0.183*"15"
+ 0.181*"8" + 0.172*"point" + 0.138*"instruct" + 0.136*"question"'), (5, '0.331*"rifl" + -0.284*"hunt" + -0.267*"money" + -0.26
7*"order" + 0.239*"load" + 0.185*"think" + 0.182*"use" + 0.173*"self" + 0.173*"deer" + 0.164*"hunter"'), (6, '-0.305*"instruct"
+ -0.267*"clock" + -0.232*"2" + 0.219*"order" + 0.219*"money" + -0.186*"point" + -0.184*"8" + -0.183*"typic" + 0.178*"rifl" + -
0.177*"would"')]
```

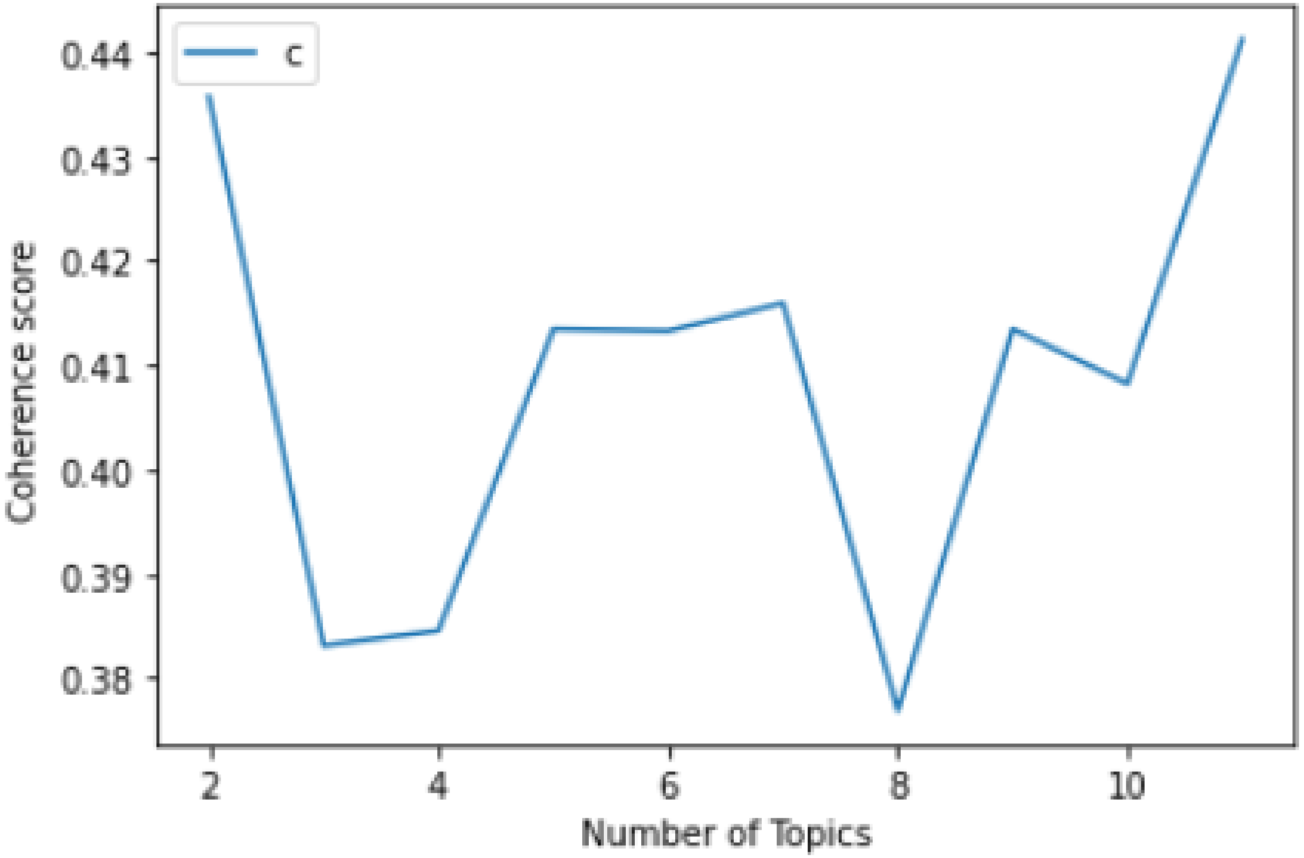
Latent Semantic Analysis (LSA): code example



```
start,stop,step=2,12,1
plot_graph(clean_text,start,stop,step)
```

Output:

Coherence score of each Topic



Coherence Score

Coherence score:

- To measure how interpretable the topics are to humans.
- Topics are represented as the top $\boldsymbol{N}$ words with the highest probability of belonging to that particular topic.
- It calculates how often two words w_i and w_j appear together in the corpus.

UMass Coherence score

$$C_{UMass}(w_i, w_j) = \log \frac{D(w_i, w_j) + 1}{D(w_i)}$$

$\boldsymbol{w_i, w_j}$: word_i and word_j

$\boldsymbol{D(w_i, w_j)}$: how many times w_i and w_j appear together.

$\boldsymbol{D(w_i)}$: how many times w_i appear alone

Note: This measurement is not symmetric, which means that $C_U Mass(w_i, w_j) \neq C_U Mass(w_j, w_i)$

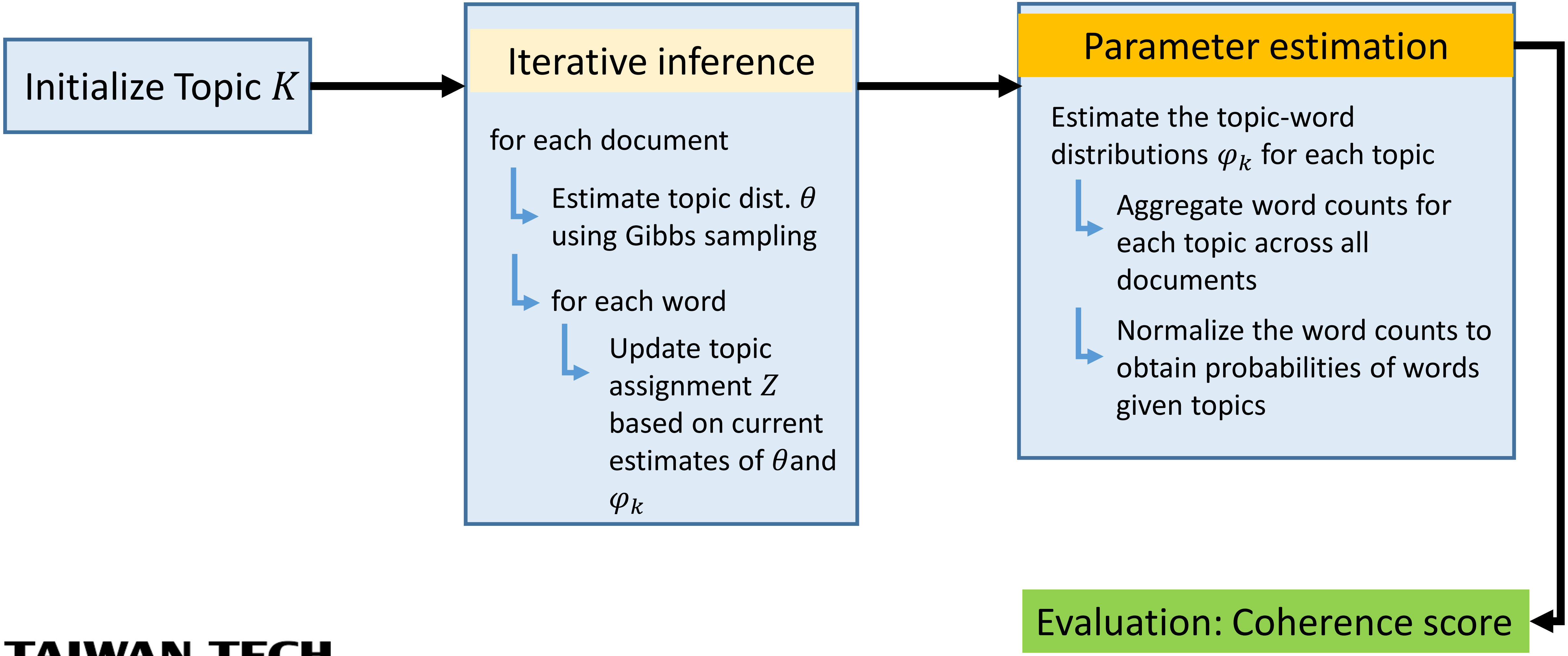
Latent Dirichlet Allocation (LDA)

By David M, Andrew Ng, Michael J, 2003

Idea → Every document is a mixture of Topics → Every topic is a mixture of words

Finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document.

Latent Dirichlet Allocation (LDA): Pipeline



Initialize Topic K

Iterative inference

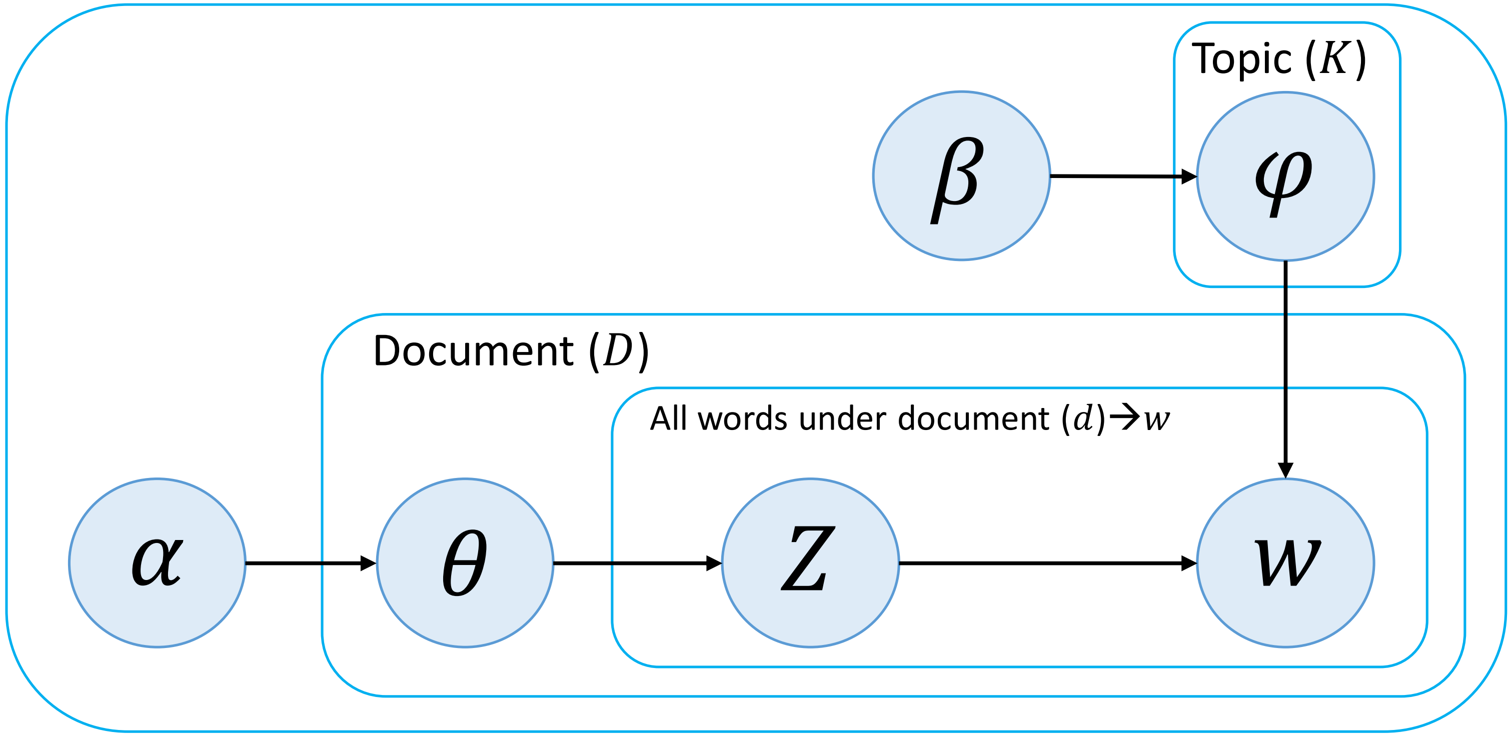
for each document
- Estimate topic dist. θ using Gibbs sampling
- for each word
  - Update topic assignment Z based on current estimates of $\theta$and φ_k

Parameter estimation

Estimate the topic-word distributions φ_k for each topic
- Aggregate word counts for each topic across all documents
- Normalize the word counts to obtain probabilities of words given topics

Evaluation: Coherence score

Latent Dirichlet Allocation (LDA): Pipeline



$\uparrow \alpha$ Lead the document to be similar in terms of what topics they contain

wider range topic

$\uparrow \beta$ Lead the topic to be similar in terms of what words they contain.

wider range of vocabulary

α : probability on the per-document topic distribution
θ_m : the distribution for document d
Z_{mn} : the topic for the n^{th} word in document d
w_{mn} : the specific word
β : probability on the per-topic word distribution
φ_k : the word distribution for topic t

Latent Dirichlet Allocation (LDA): Objective

Objective

To maximize the joint probability of observing the documents and the latent variables

$$P(W, Z, \theta, \phi; \alpha, \beta) = \prod_{i=1}^{D} P(\theta_j; \alpha) \prod_{i=1}^{K} P(\phi; \beta) \prod_{t=1}^{N} P(Z_{j,t} | \theta_j) P\left(W_{j,t} | \phi_{Z_{j,t}}\right)$$

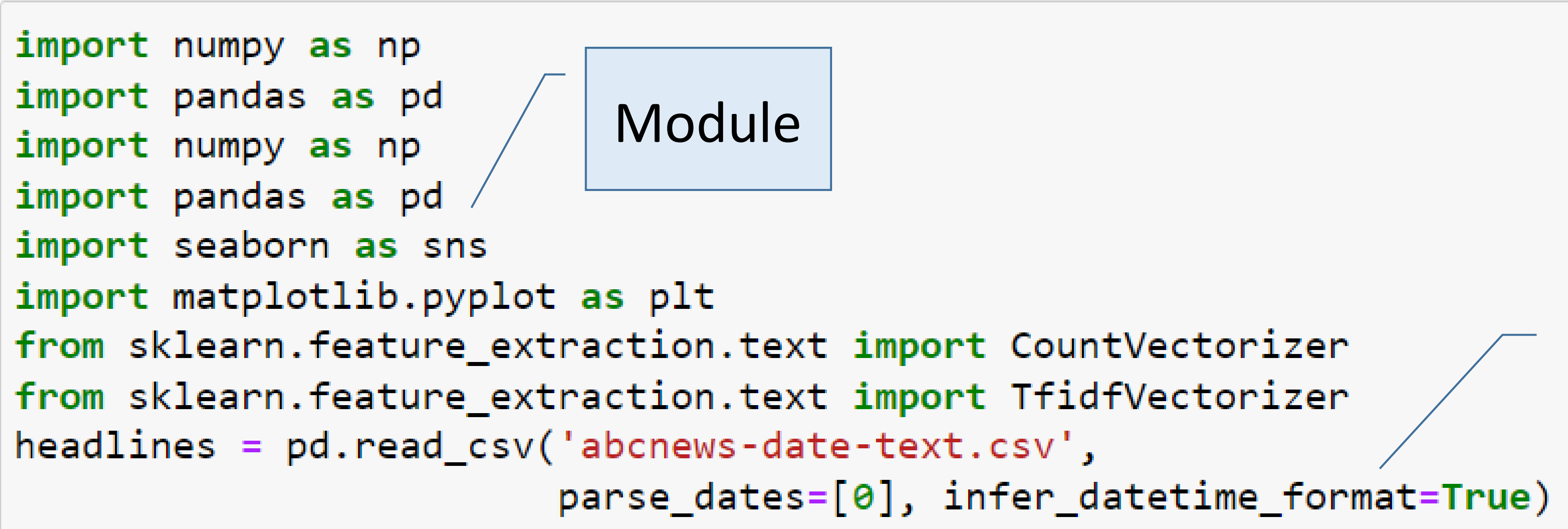
Where

α and β define Dirichlet distributions,
θ and ϕ define multinomial distributions,
Z is the vector with topics of all words in all documents,
W is the vector with all words in all documents,

D number of documents,
K number of topics and
N number of words.

Latent Dirichlet Allocation (LDA): code example

Dataset: https://www.kaggle.com/datasets/therohk/million-headlines

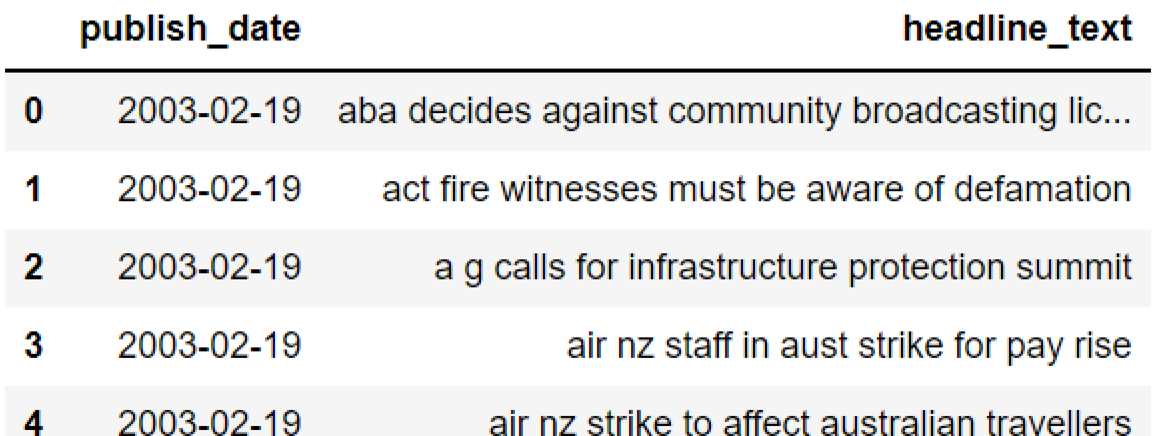


```
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
headlines = pd.read_csv('abcnews-date-text.csv',
                        parse_dates=[0], infer_datetime_format=True)
headlines.head()
```

Module

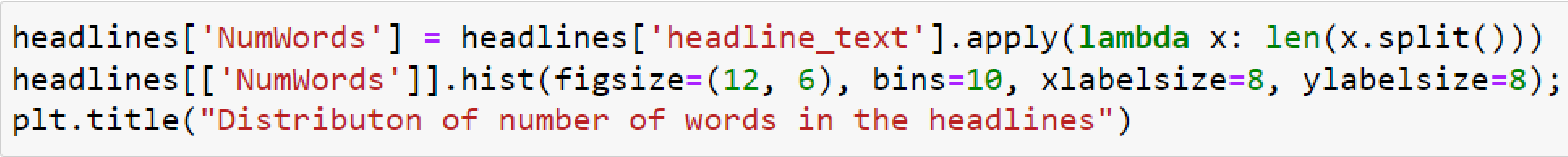
Import the data

Output



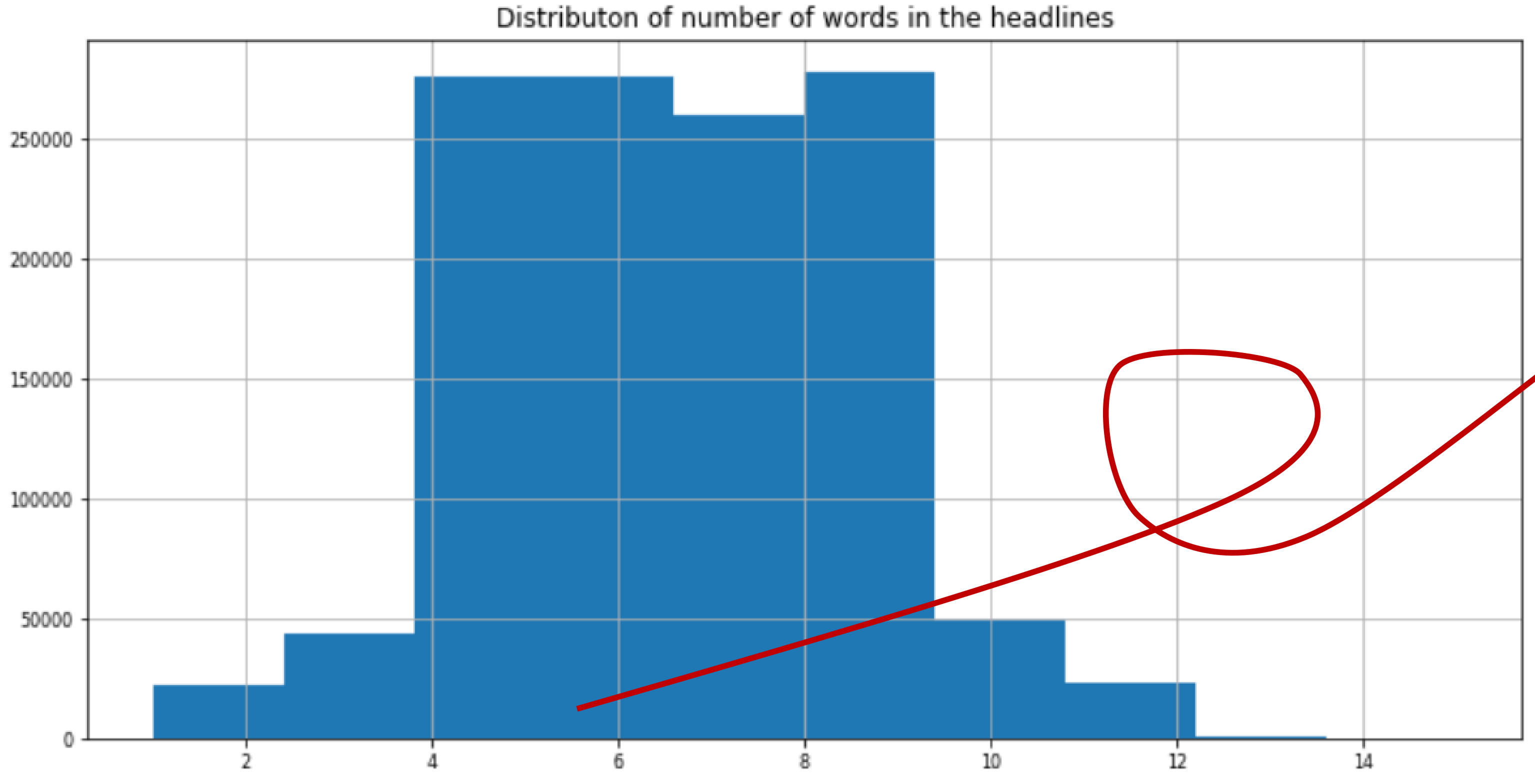
	publish_date	headline_text
0	2003-02-19	aba decides against community broadcasting lic...
1	2003-02-19	act fire witnesses must be aware of defamation
2	2003-02-19	a g calls for infrastructure protection summit
3	2003-02-19	air nz staff in aust strike for pay rise
4	2003-02-19	air nz strike to affect australian travellers

Latent Dirichlet Allocation (LDA): EDA



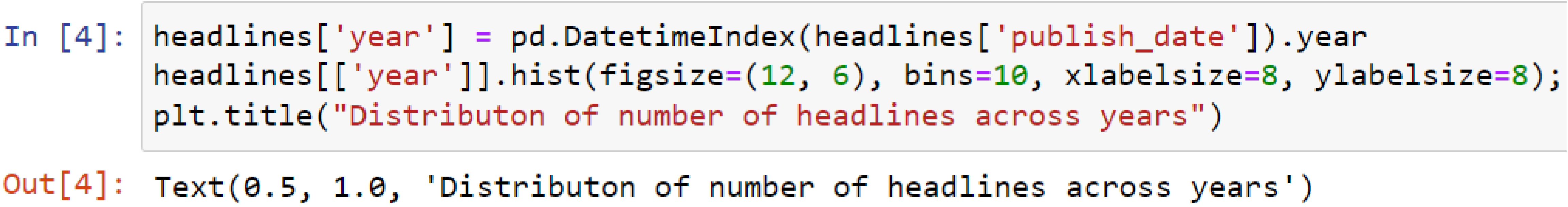
```
headlines['NumWords'] = headlines['headline_text'].apply(lambda x: len(x.split()))
headlines[['NumWords']].hist(figsize=(12, 6), bins=10, xlabelsize=8, ylabelsize=8);
plt.title("Distributon of number of words in the headlines")
```

```
Text(0.5, 1.0, 'Distributon of number of words in the headlines')
```

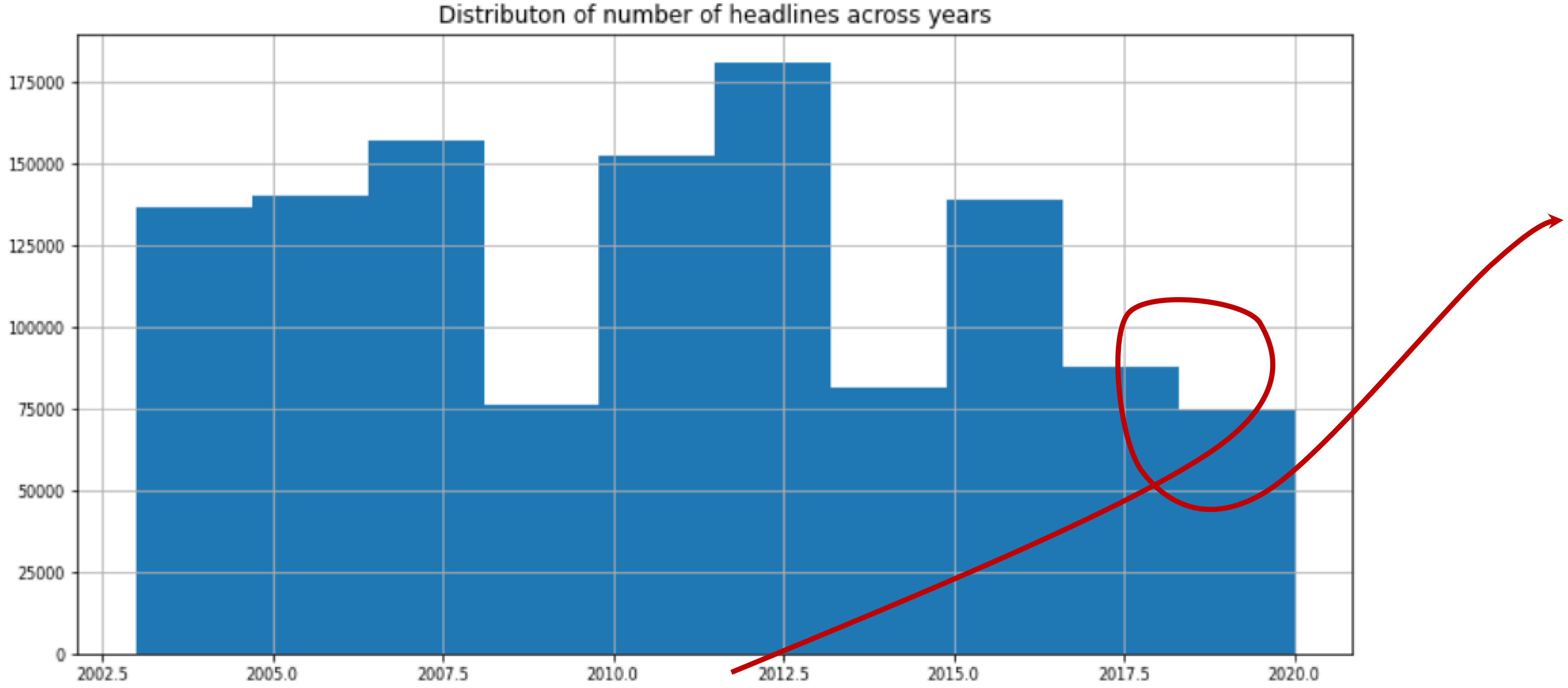


∴ The most headline have around 4~6 words.

Latent Dirichlet Allocation (LDA): EDA

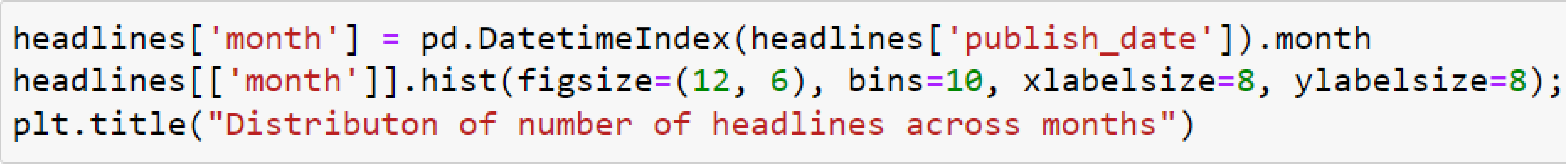


```
In [4]: headlines['year'] = pd.DatetimeIndex(headlines['publish_date']).year
        headlines[['year']].hist(figsize=(12, 6), bins=10, xlabelsize=8, ylabelsize=8);
        plt.title("Distributon of number of headlines across years")

Out[4]: Text(0.5, 1.0, 'Distributon of number of headlines across years')
```



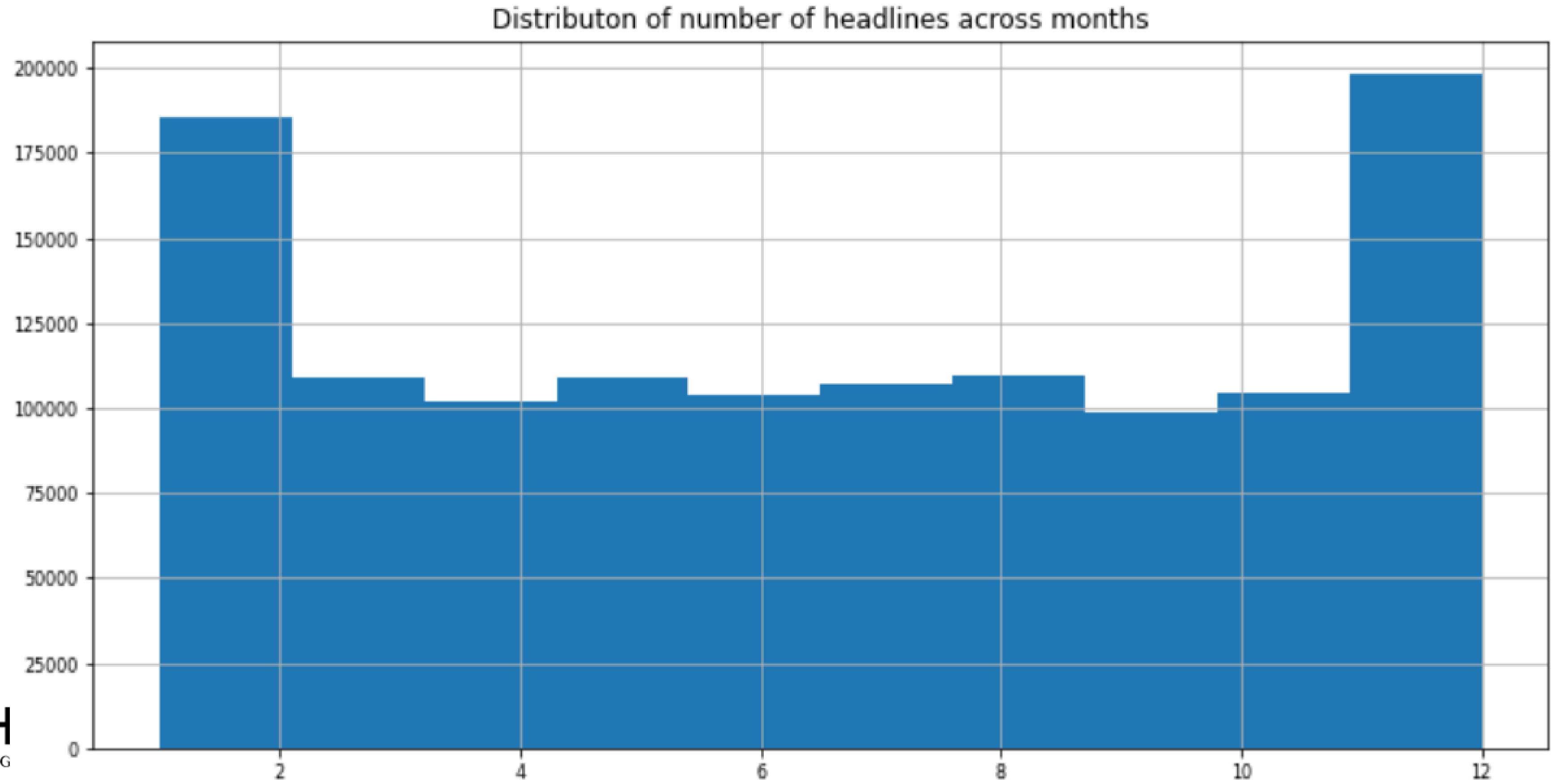
∴ 2011~2013 contribute the most of the headlines.

Latent Dirichlet Allocation (LDA): EDA



```
headlines['month'] = pd.DatetimeIndex(headlines['publish_date']).month
headlines[['month']].hist(figsize=(12, 6), bins=10, xlabelsize=8, ylabelsize=8);
plt.title("Distributon of number of headlines across months")
```

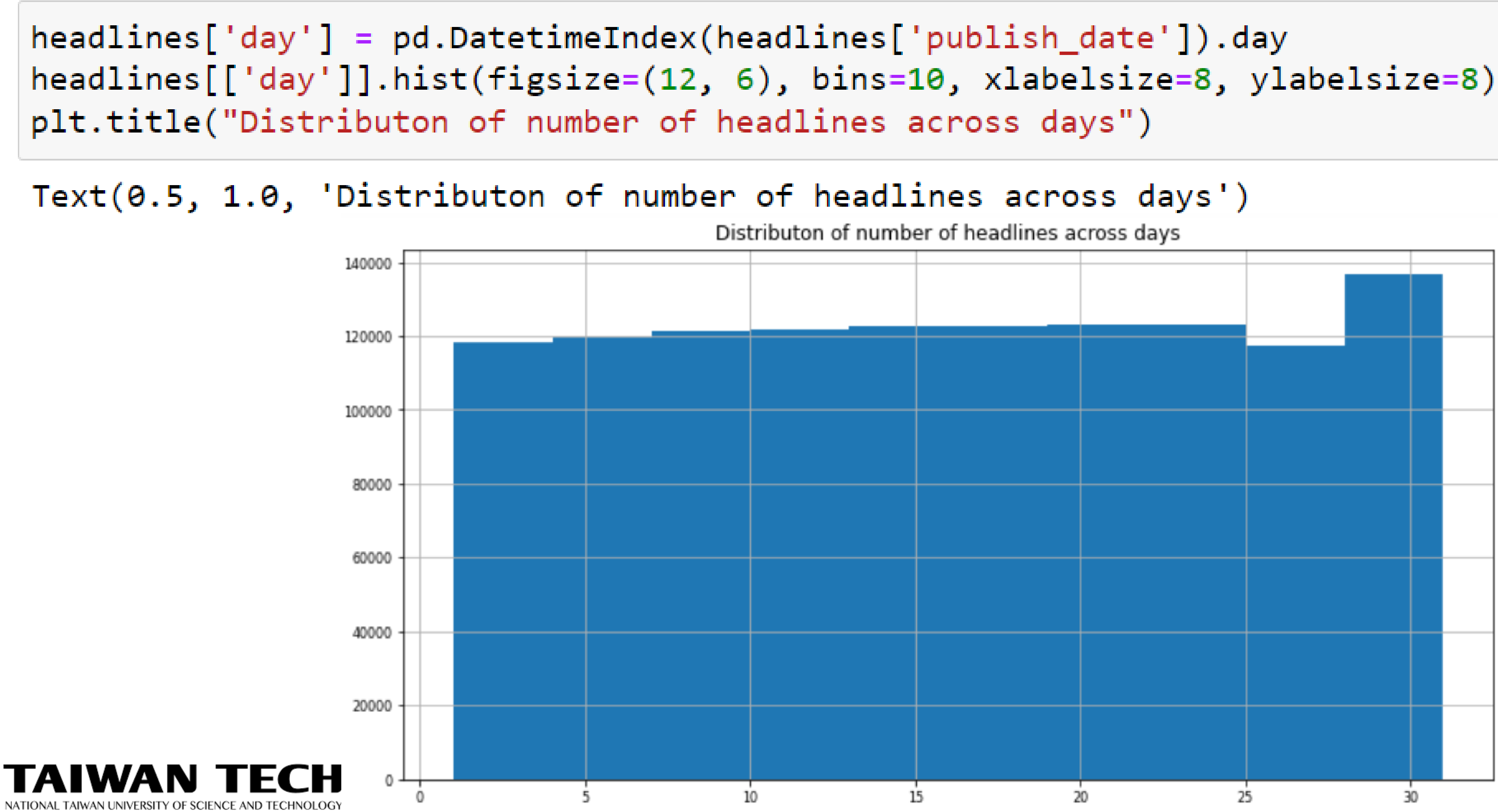
```
Text(0.5, 1.0, 'Distributon of number of headlines across months')
```



Distributon of number of headlines across months

NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOG

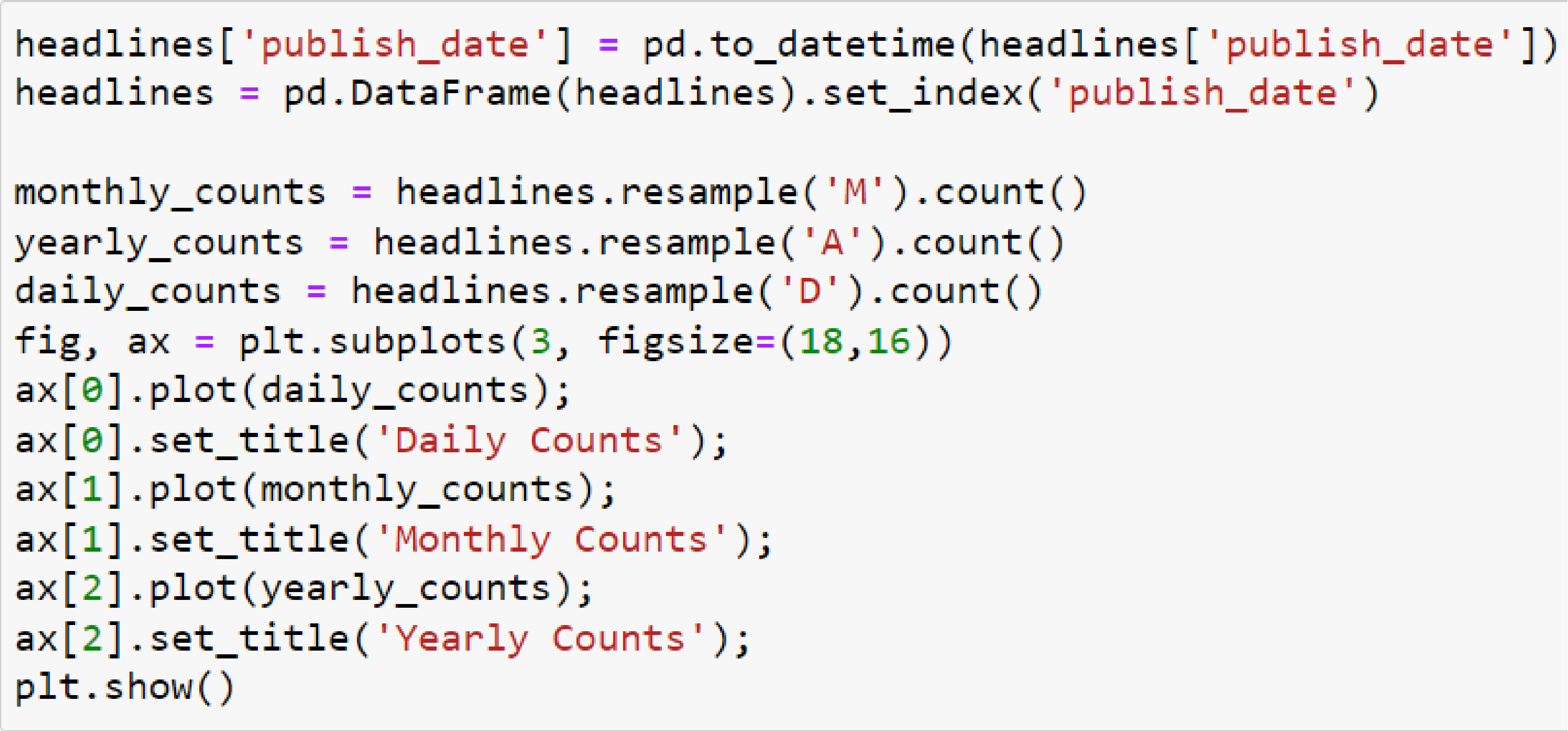
Latent Dirichlet Allocation (LDA): EDA



```
headlines['day'] = pd.DatetimeIndex(headlines['publish_date']).day
headlines[['day']].hist(figsize=(12, 6), bins=10, xlabelsize=8, ylabelsize=8);
plt.title("Distributon of number of headlines across days")
```

```
Text(0.5, 1.0, 'Distributon of number of headlines across days')
```

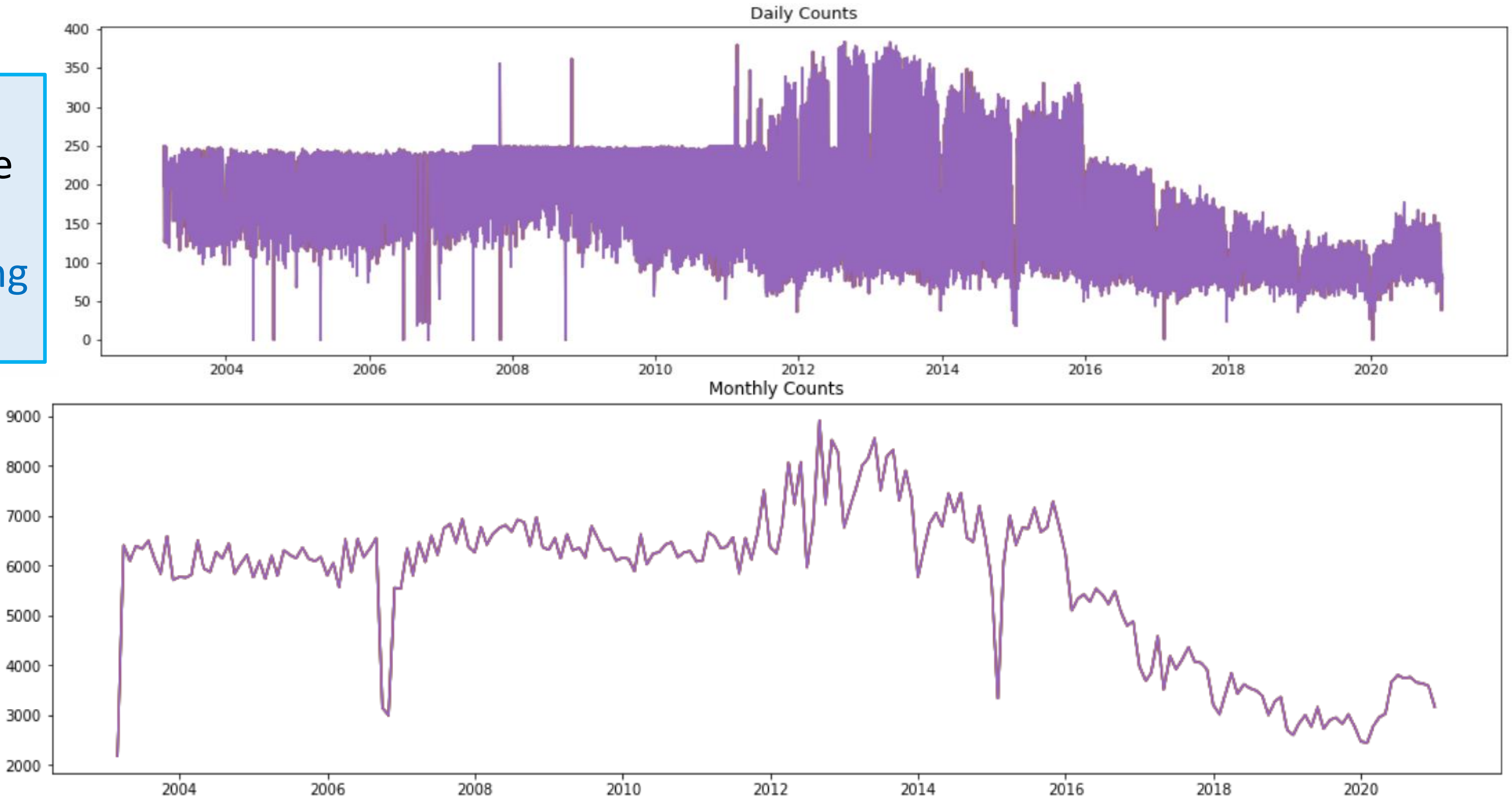
Latent Dirichlet Allocation (LDA): EDA



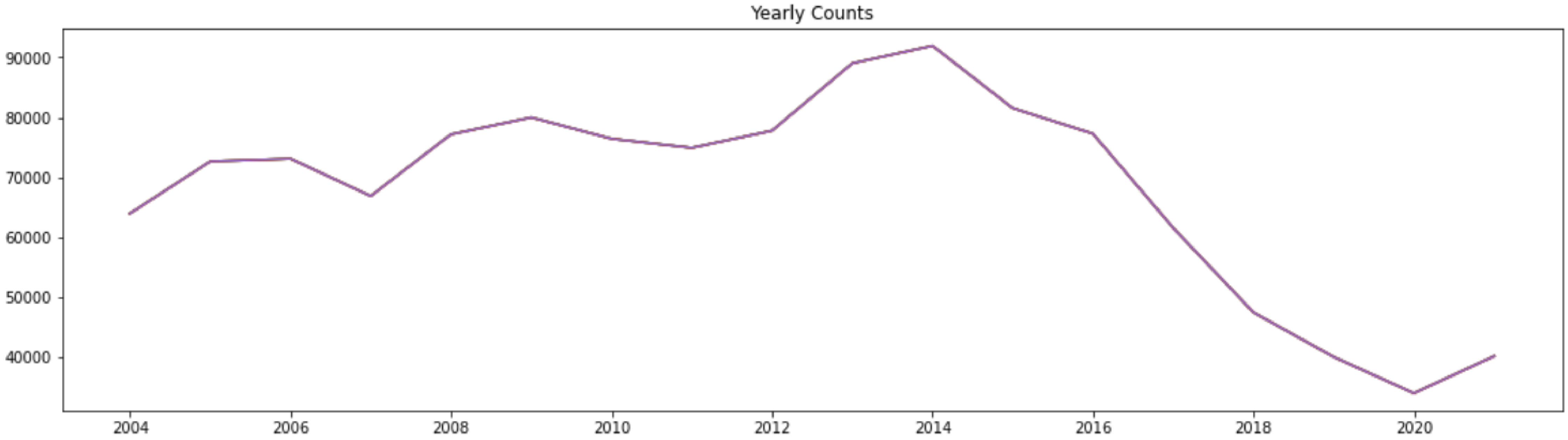
```
headlines['publish_date'] = pd.to_datetime(headlines['publish_date'])
headlines = pd.DataFrame(headlines).set_index('publish_date')

monthly_counts = headlines.resample('M').count()
yearly_counts = headlines.resample('A').count()
daily_counts = headlines.resample('D').count()
fig, ax = plt.subplots(3, figsize=(18,16))
ax[0].plot(daily_counts);
ax[0].set_title('Daily Counts');
ax[1].plot(monthly_counts);
ax[1].set_title('Monthly Counts');
ax[2].plot(yearly_counts);
ax[2].set_title('Yearly Counts');
plt.show()
```

Latent Dirichlet Allocation (LDA): EDA

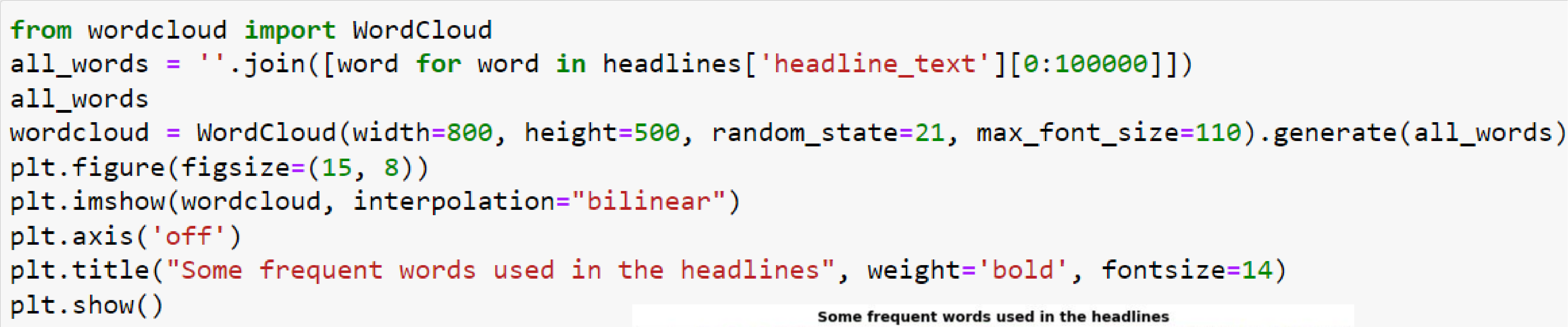
Everyday changes in the number of headlines, using time series.



Latent Dirichlet Allocation (LDA): EDA



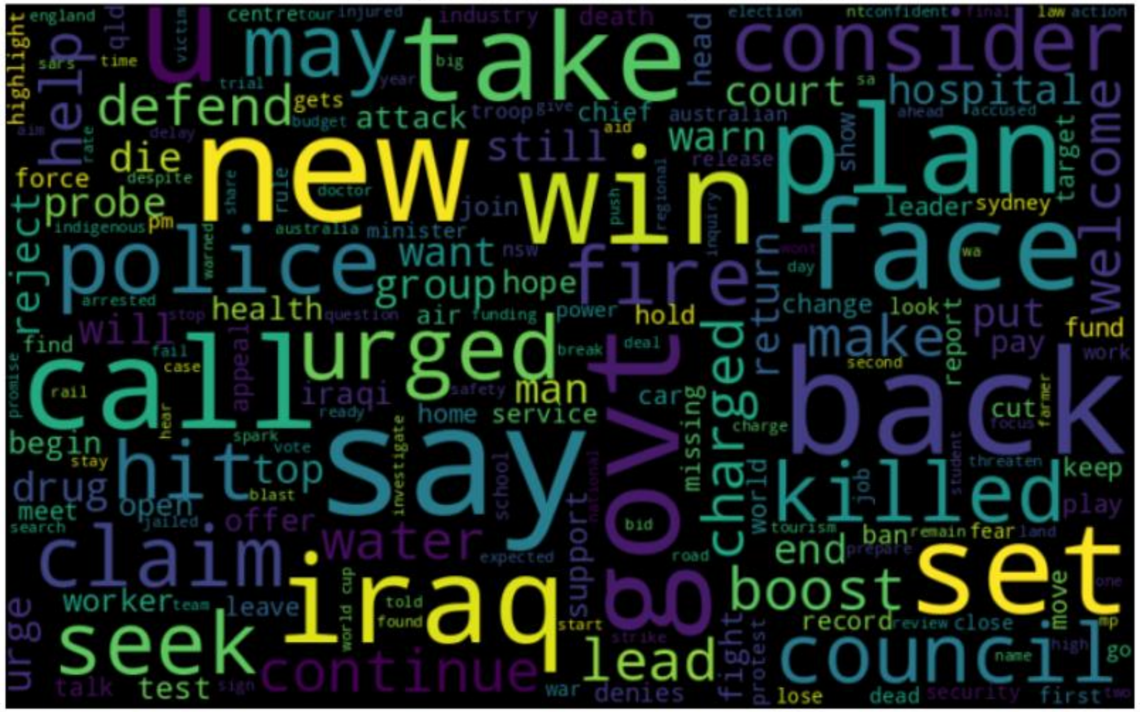
Latent Dirichlet Allocation (LDA): EDA



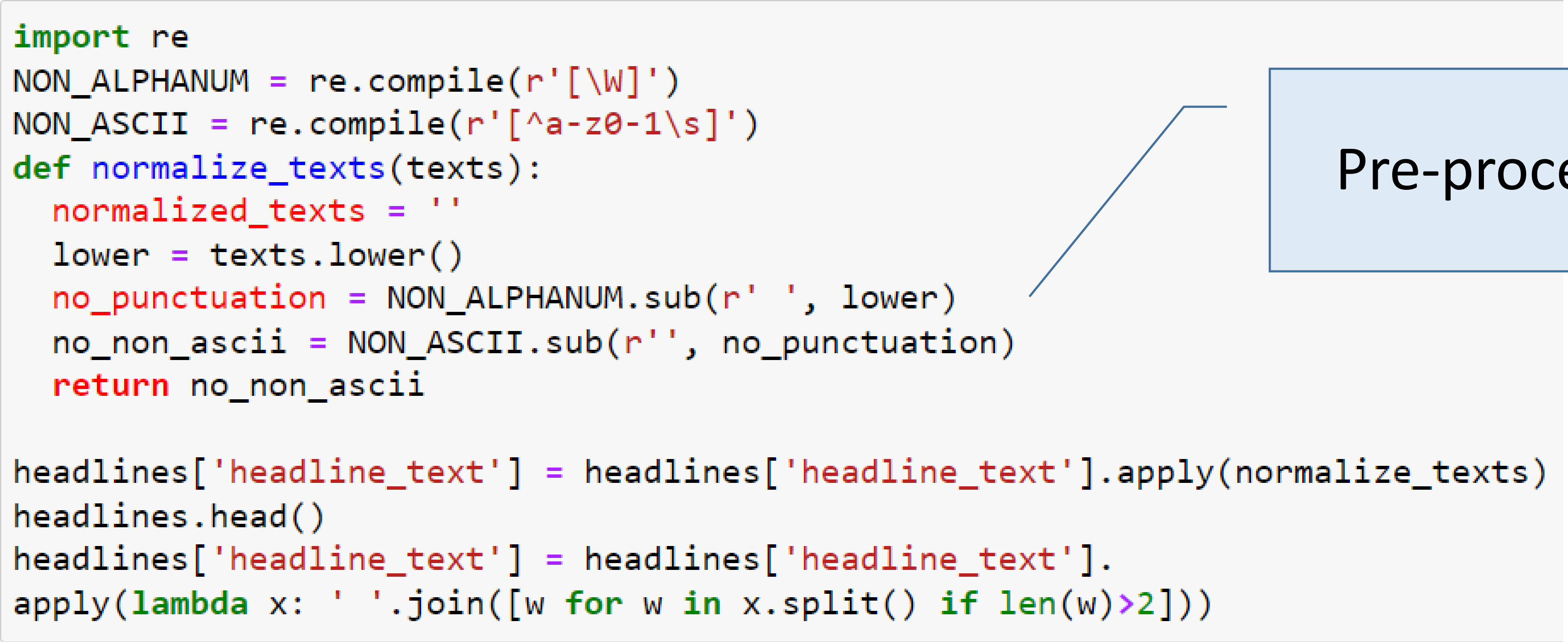
```
from wordcloud import WordCloud
all_words = ''.join([word for word in headlines['headline_text'][0:100000]])
all_words
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)
plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title("Some frequent words used in the headlines", weight='bold', fontsize=14)
plt.show()
```

The most frequent words, using word cloud.

Some frequent words used in the headlines



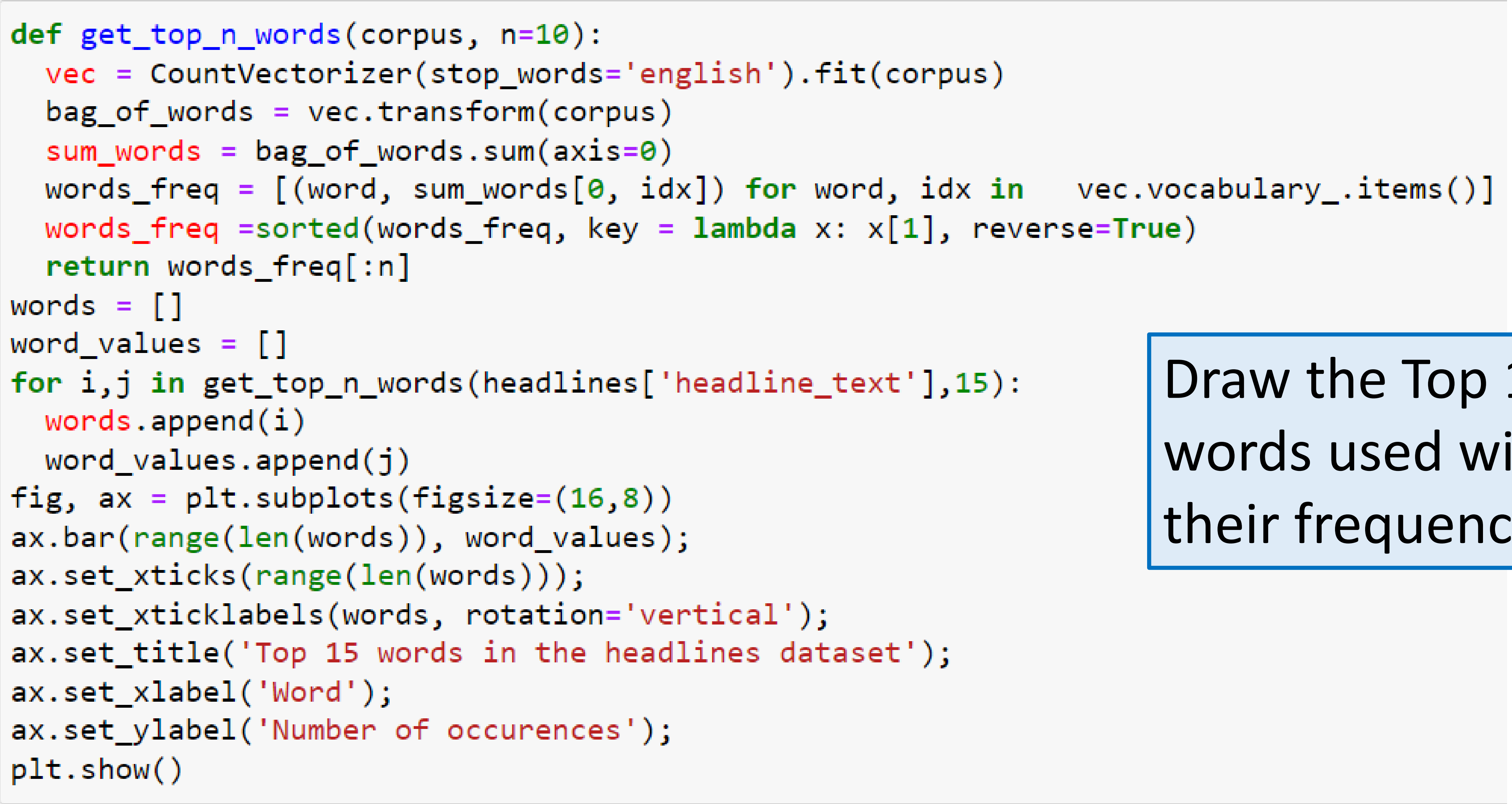
Latent Dirichlet Allocation (LDA): code example



```
import re
NON_ALPHANUM = re.compile(r'[\W]')
NON_ASCII = re.compile(r'[^a-z0-1\s]')
def normalize_texts(texts):
  normalized_texts = ''
  lower = texts.lower()
  no_punctuation = NON_ALPHANUM.sub(r' ', lower)
  no_non_ascii = NON_ASCII.sub(r'', no_punctuation)
  return no_non_ascii

headlines['headline_text'] = headlines['headline_text'].apply(normalize_texts)
headlines.head()
headlines['headline_text'] = headlines['headline_text'].
apply(lambda x: ' '.join([w for w in x.split() if len(w)>2]))
```

Pre-processing

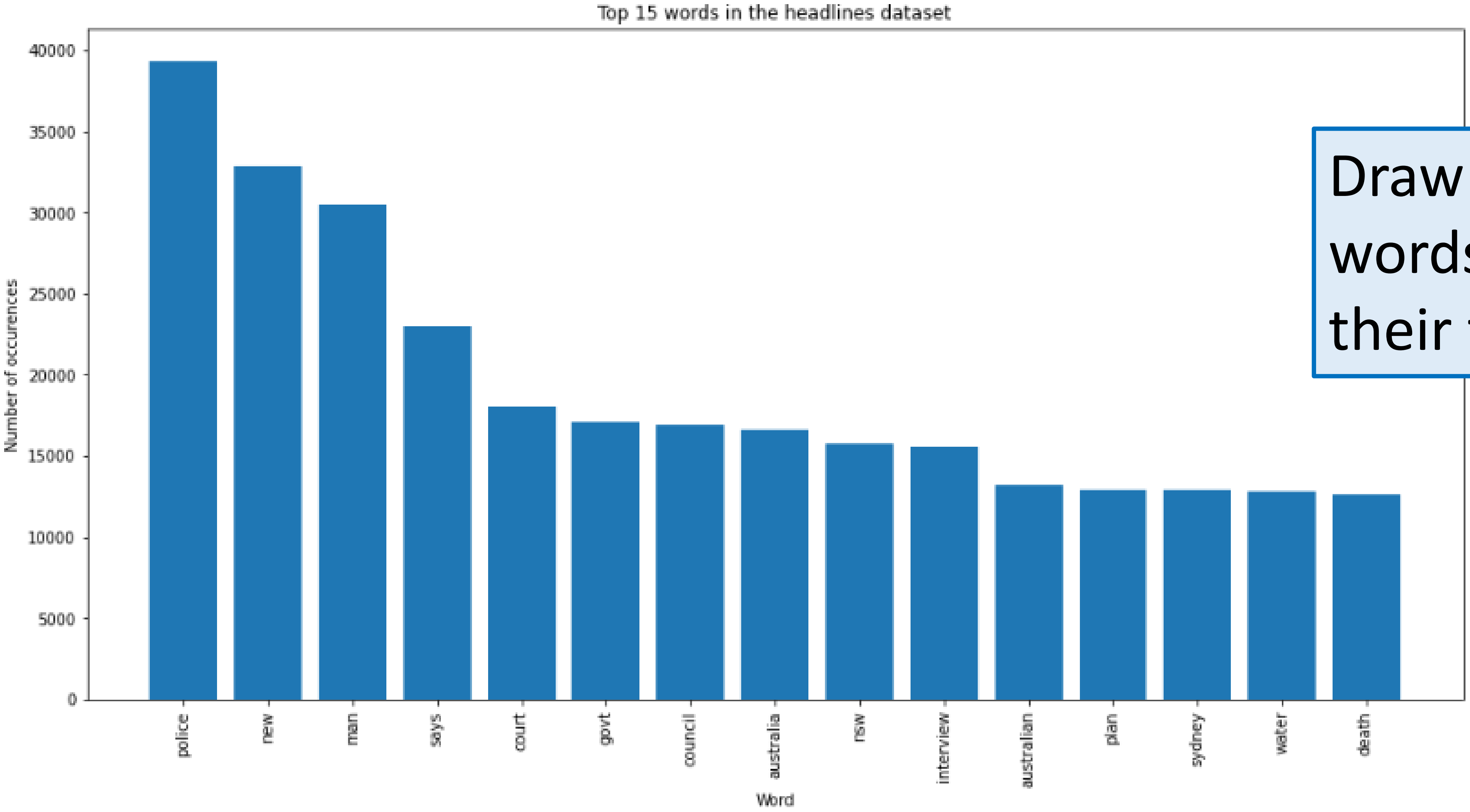
Latent Dirichlet Allocation (LDA): EDA



```
def get_top_n_words(corpus, n=10):
  vec = CountVectorizer(stop_words='english').fit(corpus)
  bag_of_words = vec.transform(corpus)
  sum_words = bag_of_words.sum(axis=0)
  words_freq = [(word, sum_words[0, idx]) for word, idx in   vec.vocabulary_.items()]
  words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
  return words_freq[:n]
words = []
word_values = []
for i,j in get_top_n_words(headlines['headline_text'],15):
  words.append(i)
  word_values.append(j)
fig, ax = plt.subplots(figsize=(16,8))
ax.bar(range(len(words)), word_values);
ax.set_xticks(range(len(words)));
ax.set_xticklabels(words, rotation='vertical');
ax.set_title('Top 15 words in the headlines dataset');
ax.set_xlabel('Word');
ax.set_ylabel('Number of occurences');
plt.show()
```

Draw the Top 15 words used with their frequencies

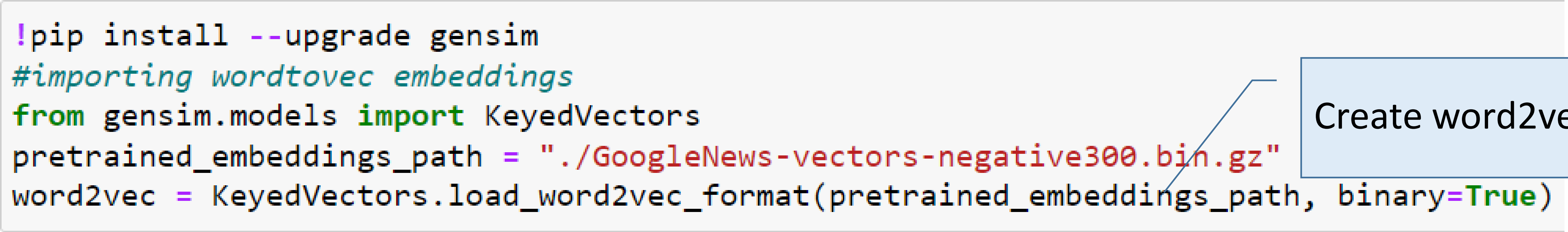
Latent Dirichlet Allocation (LDA): EDA



Draw the Top 15 words used with their frequencies

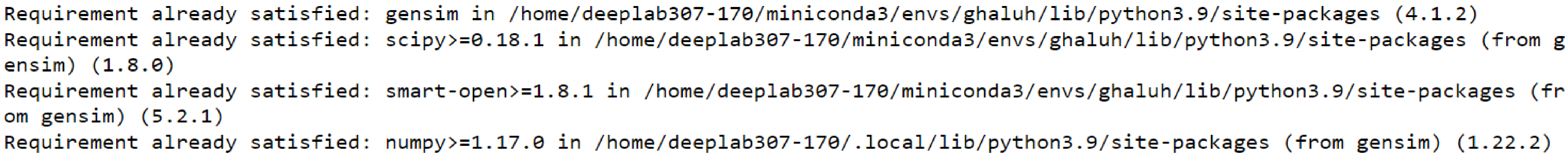
Latent Dirichlet Allocation (LDA): code example

Method 1: Clustering using word2vec embeddings



```
!pip install --upgrade gensim
#importing wordtovec embeddings
from gensim.models import KeyedVectors
pretrained_embeddings_path = "./GoogleNews-vectors-negative300.bin.gz"
word2vec = KeyedVectors.load_word2vec_format(pretrained_embeddings_path, binary=True)
```

Create word2vec

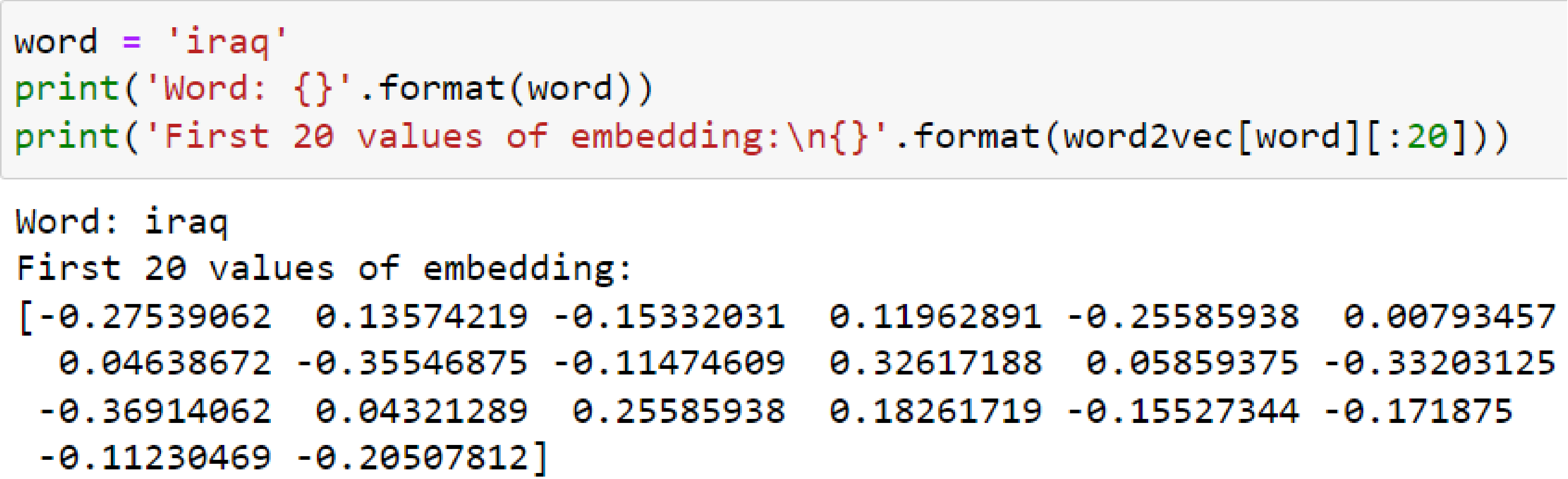


```
Requirement already satisfied: gensim in /home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages (4.1.2)
Requirement already satisfied: scipy>=0.18.1 in /home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages (from g
ensim) (1.8.0)
Requirement already satisfied: smart-open>=1.8.1 in /home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages (fr
om gensim) (5.2.1)
Requirement already satisfied: numpy>=1.17.0 in /home/deeplab307-170/.local/lib/python3.9/site-packages (from gensim) (1.22.2)
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Latent Dirichlet Allocation (LDA): code example

How word is represented in its embedding format

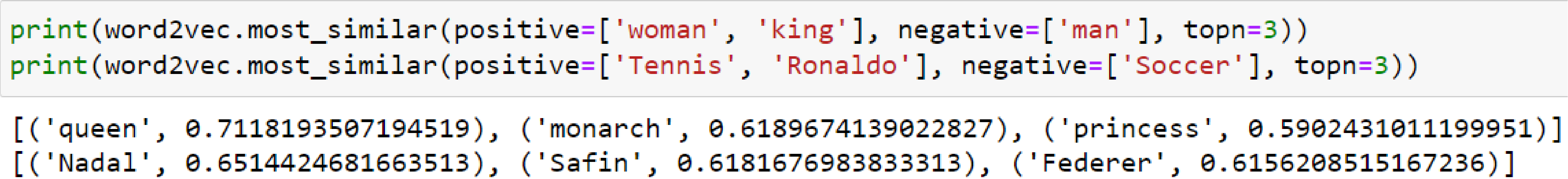


```
word = 'iraq'
print('Word: {}'.format(word))
print('First 20 values of embedding:\n{}'.format(word2vec[word][:20]))
```

```
Word: iraq
First 20 values of embedding:
[-0.27539062  0.13574219 -0.15332031  0.11962891 -0.25585938  0.00793457
  0.04638672 -0.35546875 -0.11474609  0.32617188  0.05859375 -0.33203125
 -0.36914062  0.04321289  0.25585938  0.18261719 -0.15527344 -0.171875
 -0.11230469 -0.20507812]
```

Latent Dirichlet Allocation (LDA): code example

What is the most similar answer to a word?

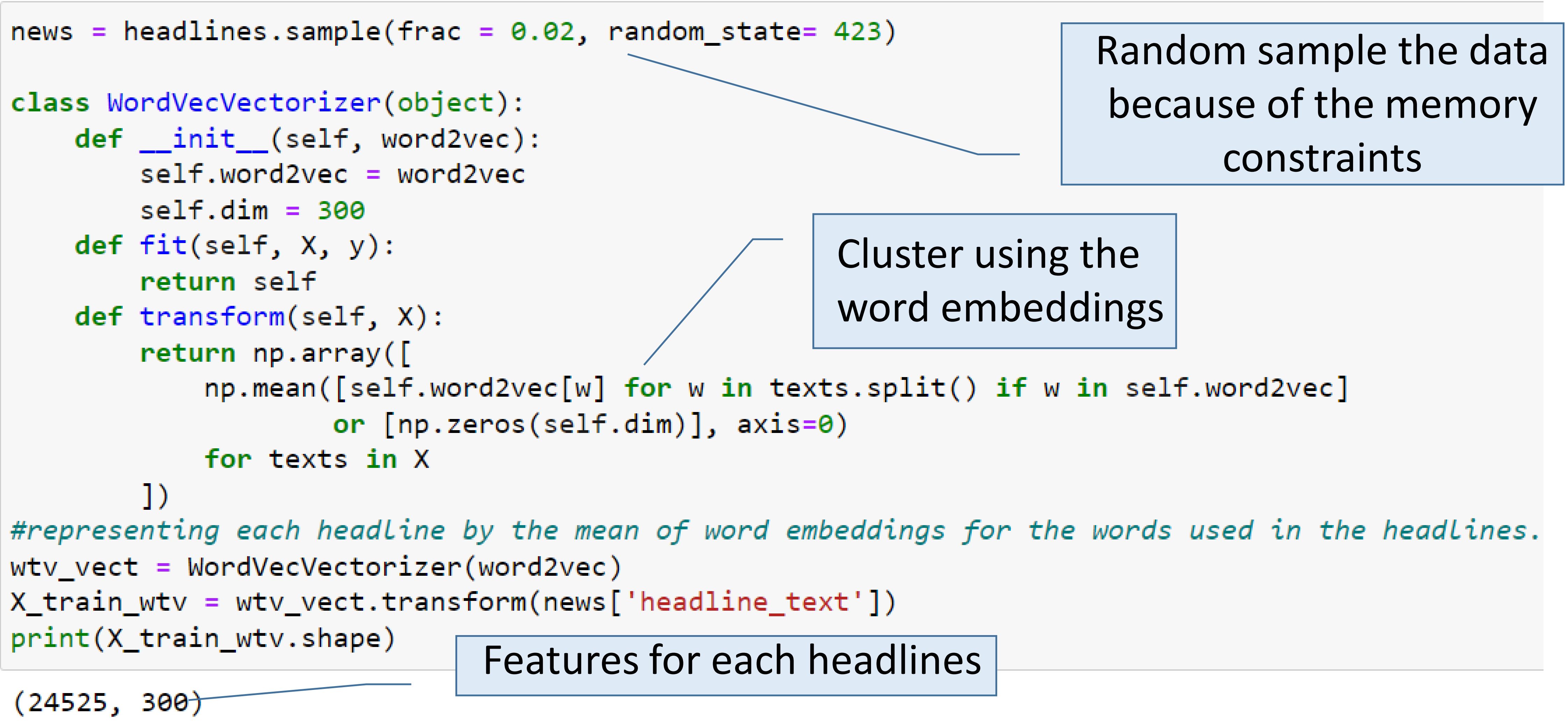


```
print(word2vec.most_similar(positive=['woman', 'king'], negative=['man'], topn=3))
print(word2vec.most_similar(positive=['Tennis', 'Ronaldo'], negative=['Soccer'], topn=3))
```

```
[('queen', 0.7118193507194519), ('monarch', 0.6189674139022827), ('princess', 0.590243101199951)]
[('Nadal', 0.651442468163513), ('Safin', 0.6181676983833313), ('Federer', 0.6156208515167236)]
```

Note:
It captures synonyms, antonyms and all the logical analogies which humans can understand.

Latent Dirichlet Allocation (LDA): code example



```
news = headlines.sample(frac = 0.02, random_state= 423)

class WordVecVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        self.dim = 300
    def fit(self, X, y):
        return self
    def transform(self, X):
        return np.array([
            np.mean([self.word2vec[w] for w in texts.split() if w in self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for texts in X
        ])
#representing each headline by the mean of word embeddings for the words used in the headlines.
wtv_vect = WordVecVectorizer(word2vec)
X_train_wtv = wtv_vect.transform(news['headline_text'])
print(X_train_wtv.shape)
```

```
(24525, 300)
```

Random sample the data because of the memory constraints

Cluster using the word embeddings

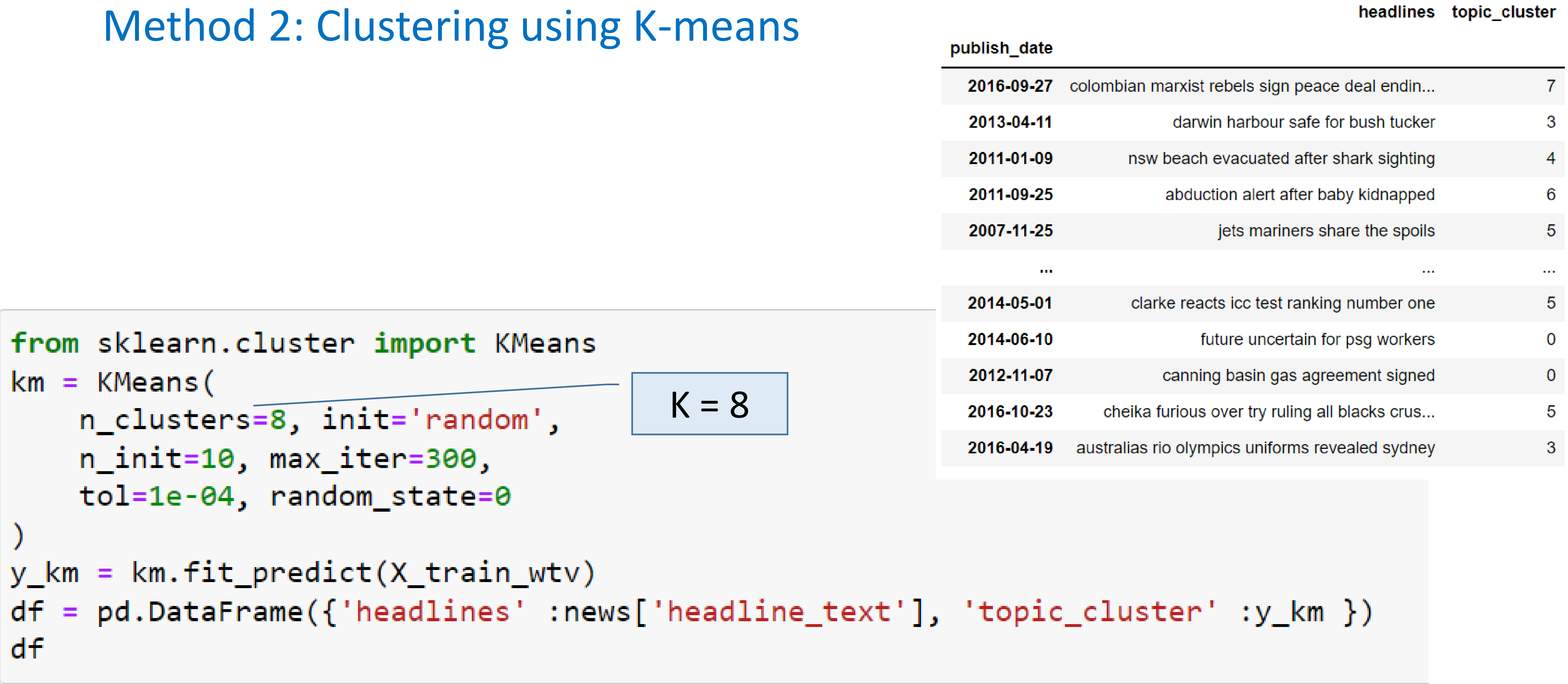
Features for each headlines

Total headlines

Source: https://numpy.org/doc/stable/reference/generated/numpy.mean.html

Latent Dirichlet Allocation (LDA): code example

Method 2: Clustering using K-means



	headlines	topic_cluster
publish_date		
2016-09-27	colombian marxist rebels sign peace deal endin...	7
2013-04-11	darwin harbour safe for bush tucker	3
2011-01-09	nsw beach evacuated after shark sighting	4
2011-09-25	abduction alert after baby kidnapped	6
2007-11-25	jets mariners share the spoils	5
...	...	...
2014-05-01	clarke reacts icc test ranking number one	5
2014-06-10	future uncertain for psg workers	0
2012-11-07	canning basin gas agreement signed	0
2016-10-23	cheika furious over try ruling all blacks crus...	5
2016-04-19	australias rio olympics uniforms revealed sydney	3

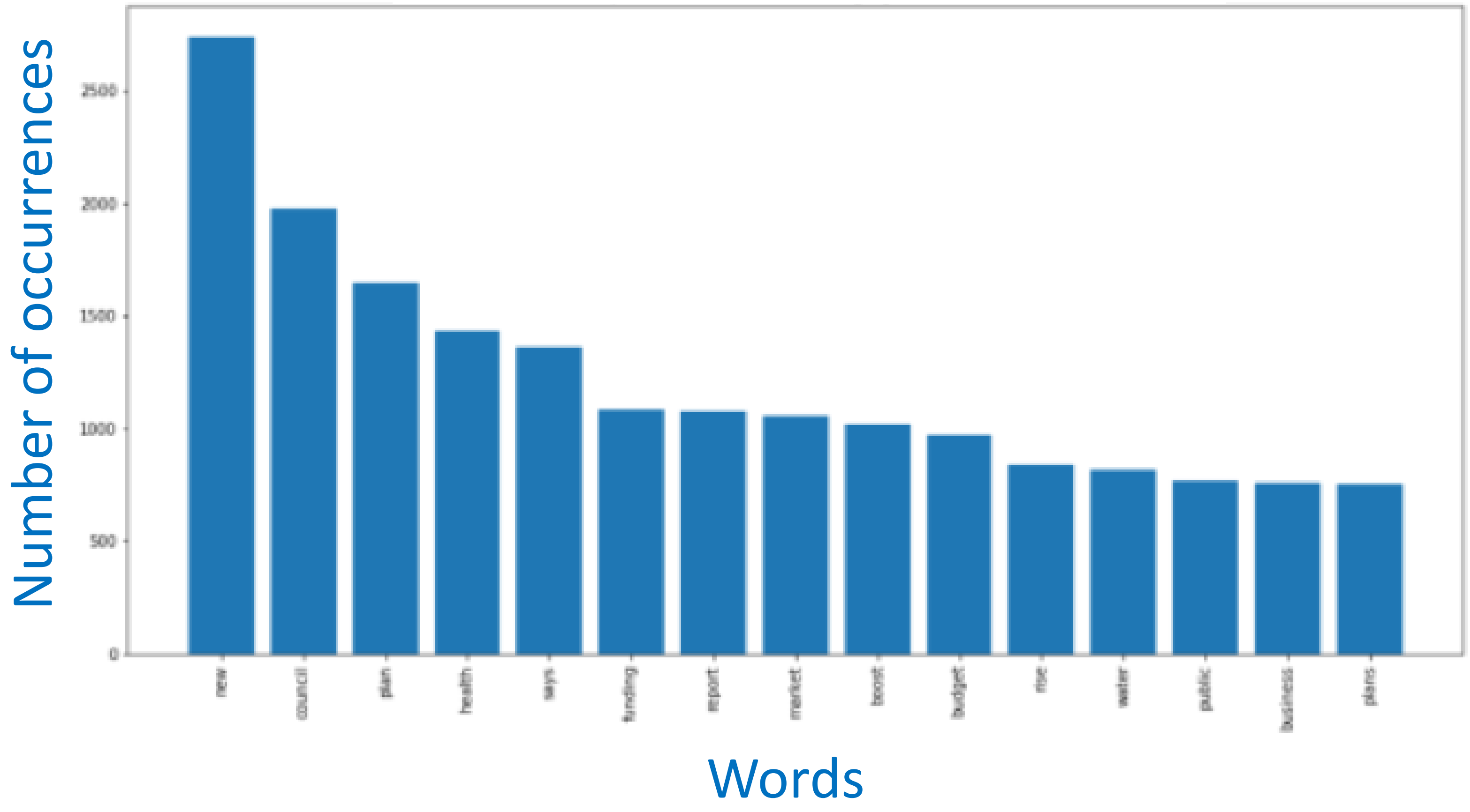
```
from sklearn.cluster import KMeans
km = KMeans(
    n_clusters=8, init='random',
    n_init=10, max_iter=300,
    tol=1e-04, random_state=0
)
y_km = km.fit_predict(X_train_wtv)
df = pd.DataFrame({'headlines' :news['headline_text'], 'topic_cluster' :y_km })
df
```

K = 8

Source: https://scikit-learn.org/stable/modules/clustering.html#k-means

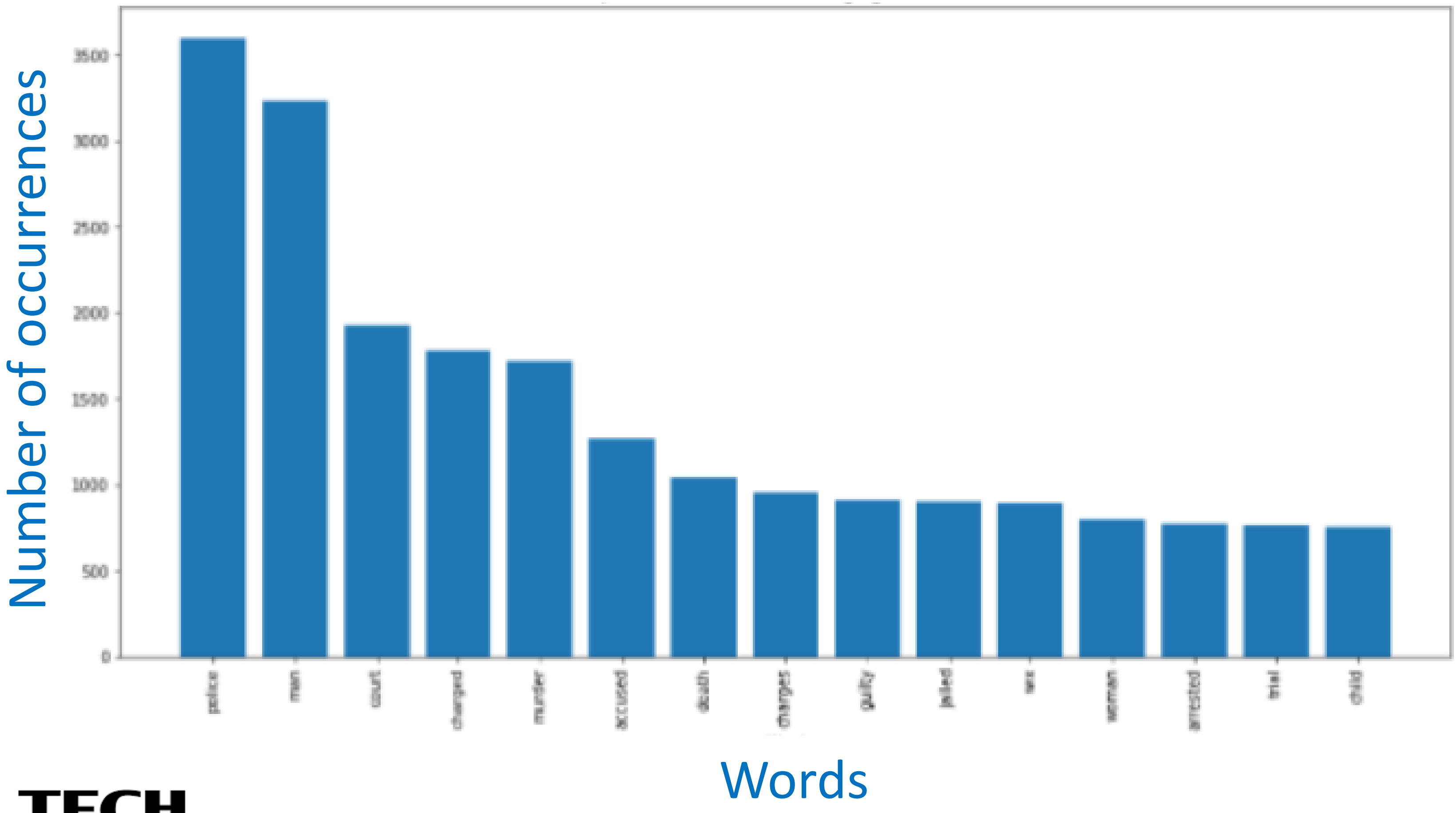
Latent Dirichlet Allocation (LDA): code example

Top 15 words belong to cluster 0



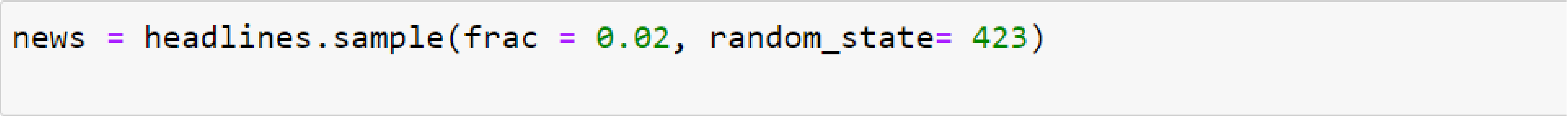
Latent Dirichlet Allocation (LDA): code example

Top 15 words belong to cluster 1

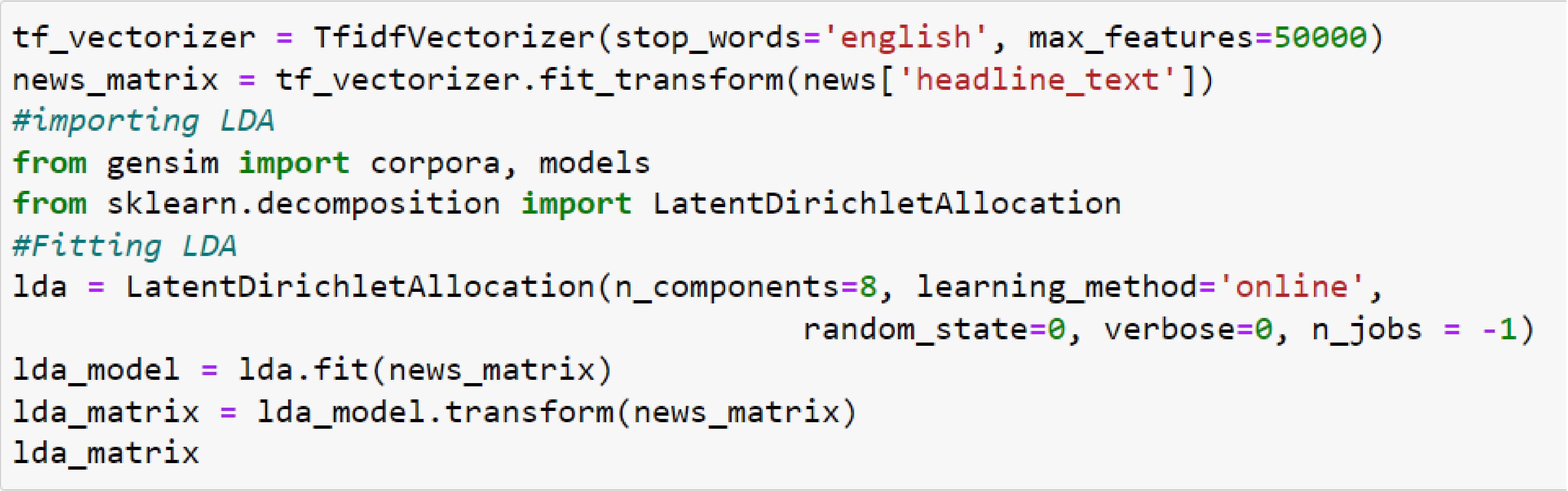


Latent Dirichlet Allocation (LDA): code example

Method 3: Topic modeling with LDA



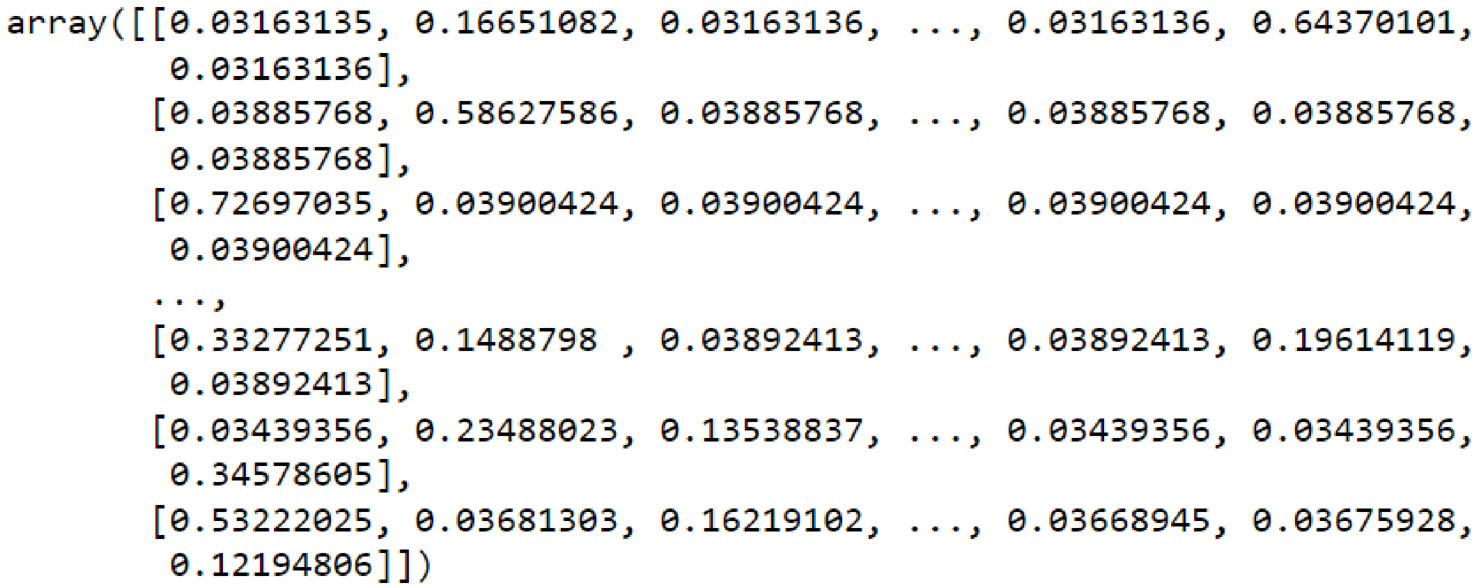
```
news = headlines.sample(frac = 0.02, random_state= 423)
```



```
tf_vectorizer = TfidfVectorizer(stop_words='english', max_features=50000)
news_matrix = tf_vectorizer.fit_transform(news['headline_text'])
#importing LDA
from gensim import corpora, models
from sklearn.decomposition import LatentDirichletAllocation
#Fitting LDA
lda = LatentDirichletAllocation(n_components=8, learning_method='online',
                                          random_state=0, verbose=0, n_jobs = -1)
lda_model = lda.fit(news_matrix)
lda_matrix = lda_model.transform(news_matrix)
lda_matrix
```

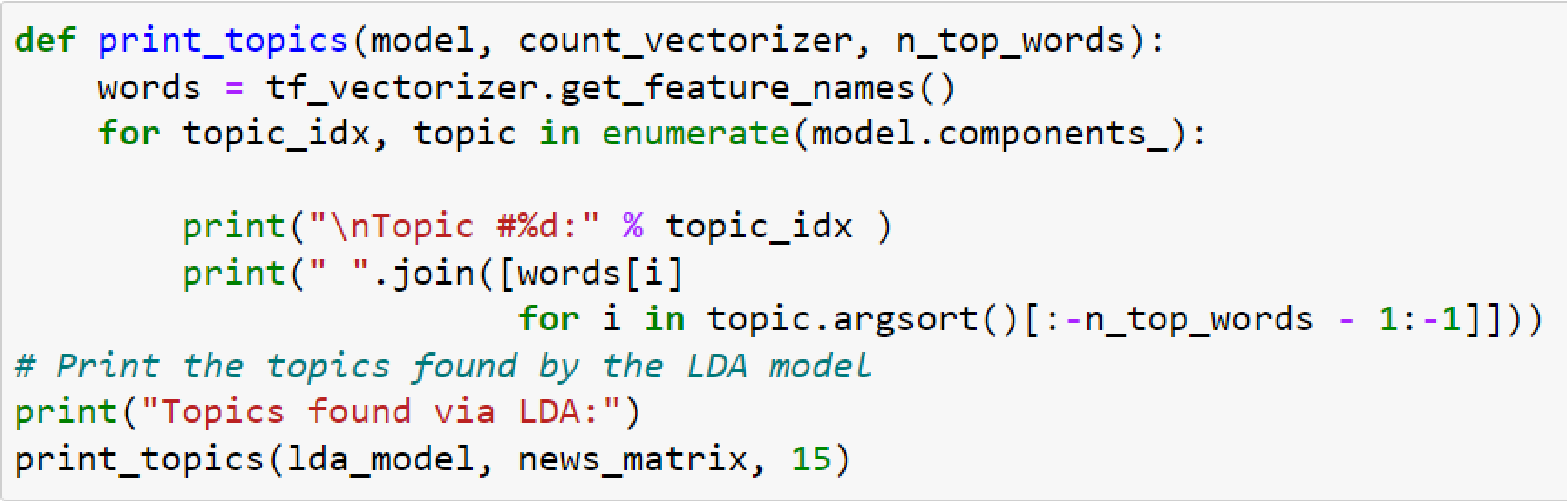
Latent Dirichlet Allocation (LDA): code example

Output



```
array([[0.03163135, 0.16651082, 0.03163136, ..., 0.03163136, 0.64370101,
        0.03163136],
       [0.03885768, 0.58627586, 0.03885768, ..., 0.03885768, 0.03885768,
        0.03885768],
       [0.72697035, 0.03900424, 0.03900424, ..., 0.03900424, 0.03900424,
        0.03900424],
       ...,
       [0.33277251, 0.1488798 , 0.03892413, ..., 0.03892413, 0.19614119,
        0.03892413],
       [0.03439356, 0.23488023, 0.13538837, ..., 0.03439356, 0.03439356,
        0.34578605],
       [0.53222025, 0.03681303, 0.16219102, ..., 0.03668945, 0.03675928,
        0.12194806]])
```

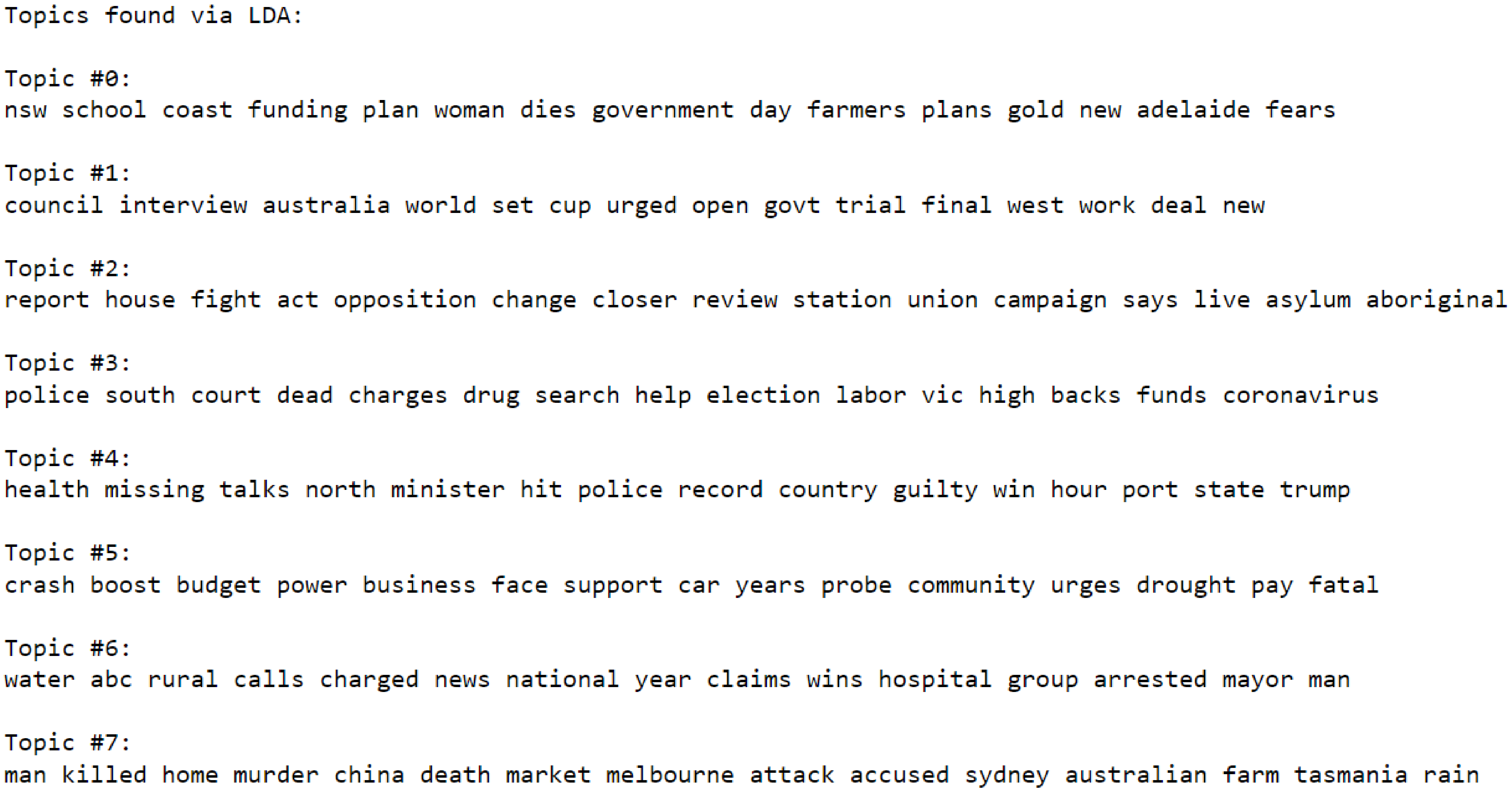
Latent Dirichlet Allocation (LDA): code example



```
def print_topics(model, count_vectorizer, n_top_words):
    words = tf_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):

        print("\nTopic #%d:" % topic_idx )
        print(" ".join([words[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))
# Print the topics found by the LDA model
print("Topics found via LDA:")
print_topics(lda_model, news_matrix, 15)
```

Latent Dirichlet Allocation (LDA): code example

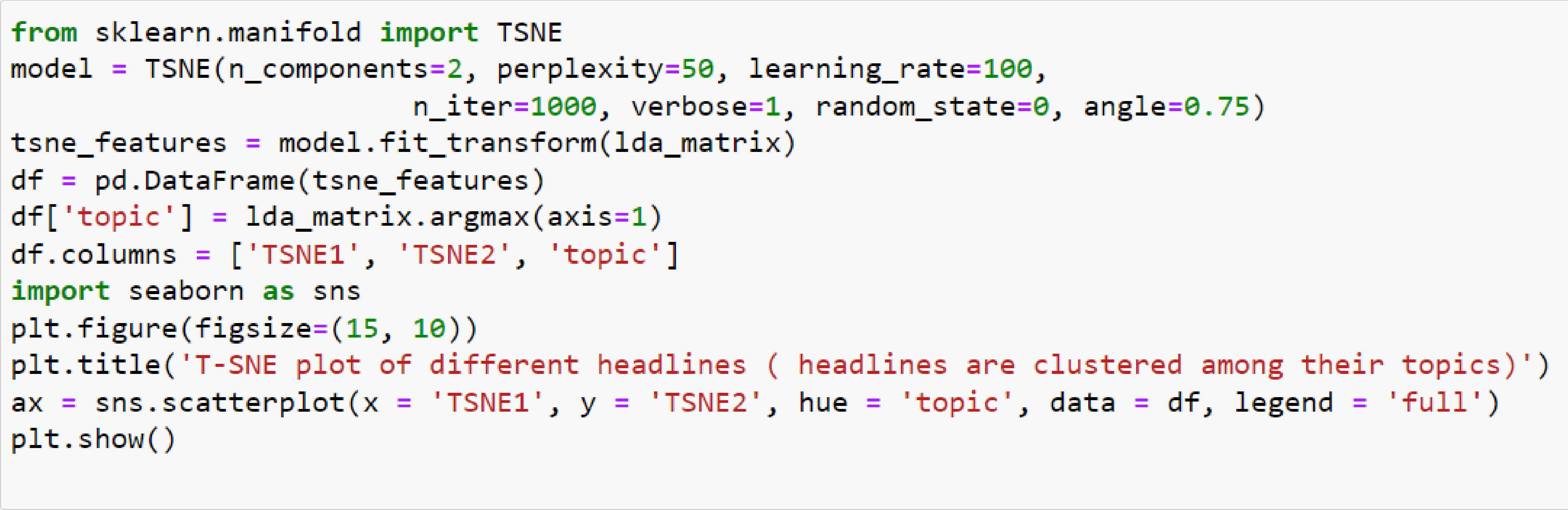
Output:



```
Topics found via LDA:

Topic #0:
nsw school coast funding plan woman dies government day farmers plans gold new adelaide fears

Topic #1:
council interview australia world set cup urged open govt trial final west work deal new

Topic #2:
report house fight act opposition change closer review station union campaign says live asylum aboriginal

Topic #3:
police south court dead charges drug search help election labor vic high backs funds coronavirus

Topic #4:
health missing talks north minister hit police record country guilty win hour port state trump

Topic #5:
crash boost budget power business face support car years probe community urges drought pay fatal

Topic #6:
water abc rural calls charged news national year claims wins hospital group arrested mayor man

Topic #7:
man killed home murder china death market melbourne attack accused sydney australian farm tasmania rain
```

Latent Dirichlet Allocation (LDA): code example

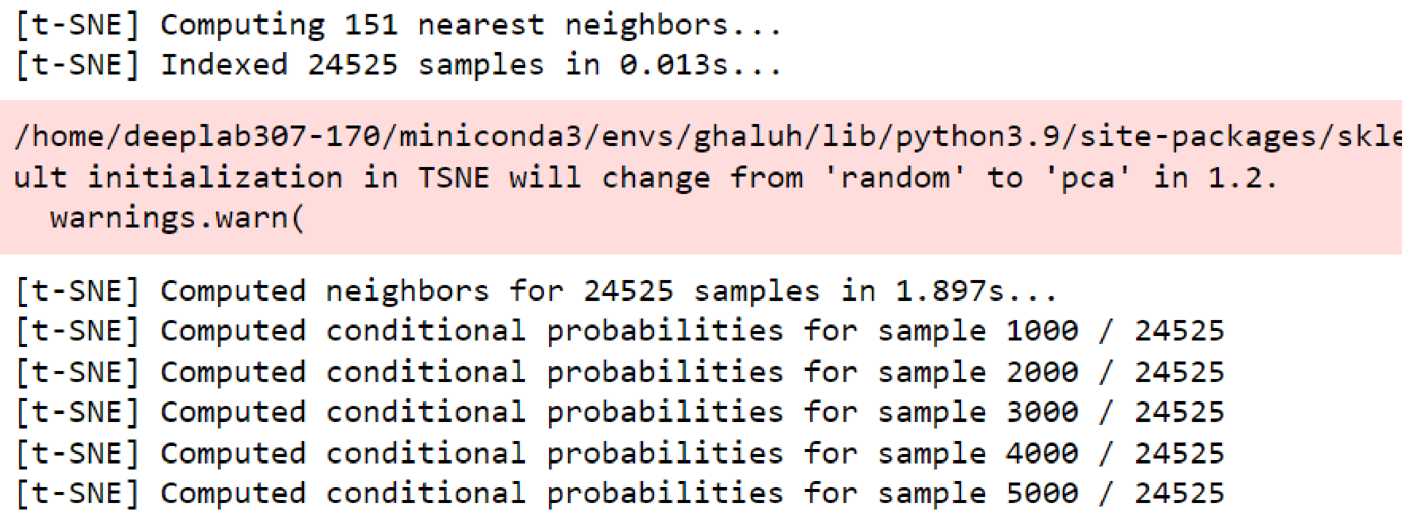
T-SNE Visualization



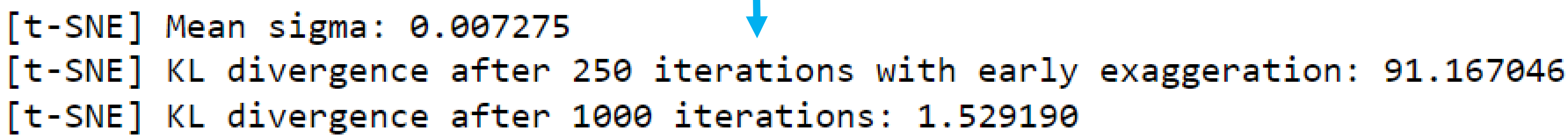
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, perplexity=50, learning_rate=100,
                        n_iter=1000, verbose=1, random_state=0, angle=0.75)
tsne_features = model.fit_transform(lda_matrix)
df = pd.DataFrame(tsne_features)
df['topic'] = lda_matrix.argmax(axis=1)
df.columns = ['TSNE1', 'TSNE2', 'topic']
import seaborn as sns
plt.figure(figsize=(15, 10))
plt.title('T-SNE plot of different headlines ( headlines are clustered among their topics)')
ax = sns.scatterplot(x = 'TSNE1', y = 'TSNE2', hue = 'topic', data = df, legend = 'full')
plt.show()
```

Latent Dirichlet Allocation (LDA): code example

Output:



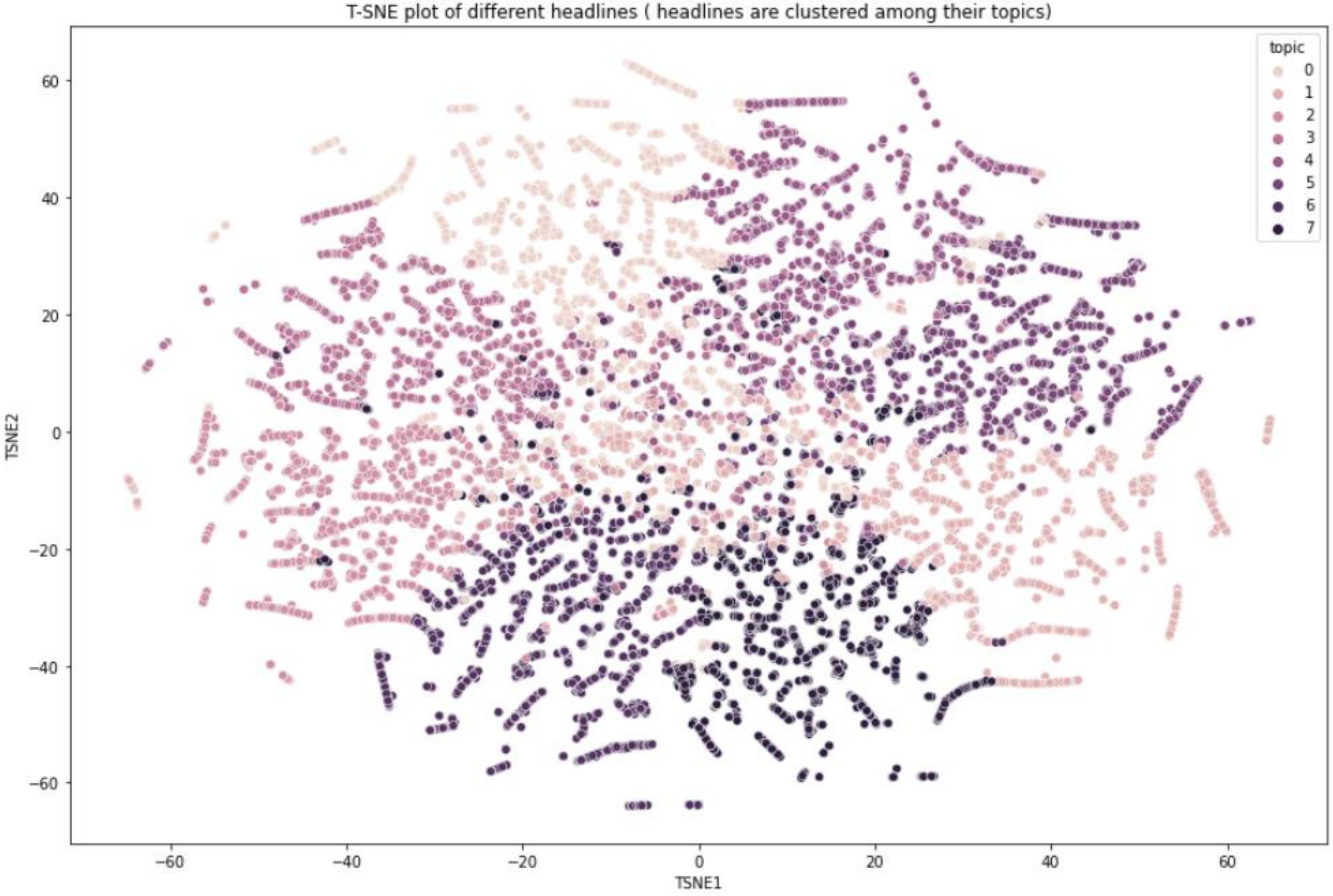
```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 24525 samples in 0.013s...

/home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages/skle
ult initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(

[t-SNE] Computed neighbors for 24525 samples in 1.897s...
[t-SNE] Computed conditional probabilities for sample 1000 / 24525
[t-SNE] Computed conditional probabilities for sample 2000 / 24525
[t-SNE] Computed conditional probabilities for sample 3000 / 24525
[t-SNE] Computed conditional probabilities for sample 4000 / 24525
[t-SNE] Computed conditional probabilities for sample 5000 / 24525
```



```
[t-SNE] Mean sigma: 0.007275
[t-SNE] KL divergence after 250 iterations with early exaggeration: 91.167046
[t-SNE] KL divergence after 1000 iterations: 1.529190
```

Latent Dirichlet Allocation (LDA): code example

Output:



Thank you
Q & A

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY