



Complete NLP in Machine Learning: An End-to-End Guide

Hands-on practice



What is NLP?

Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) that enables computers to understand, interpret, and generate human language. It combines linguistics, machine learning, and computer science to process text and speech.

Importance of NLP

- ◆ Helps computers understand human language like a human.
- ◆ Automates repetitive language-related tasks.
- ◆ Bridges the gap between humans and machines (e.g., chatbots, voice assistants).
- ◆ Extracts insights from large amounts of text data.

Real-World Applications of NLP

- ✓ **Chatbots & Virtual Assistants** – Siri, Alexa, Google Assistant
- ✓ **Machine Translation** – Google Translate
- ✓ **Speech Recognition** – Voice-to-text (YouTube captions, WhatsApp dictation)
- ✓ **Sentiment Analysis** – Identifying positive/negative reviews
- ✓ **Spam Detection** – Filtering unwanted emails
- ✓ **Text Summarization** – AI-generated summaries for news articles
- ✓ **Named Entity Recognition (NER)** – Extracting names, dates, and locations from documents

Setting Up a Python Environment for NLP with NLTK 🚀

To work with **NLTK** for **Natural Language Processing (NLP)**, follow these steps to set up your Python environment.

◆ Step 1: Install Python (If Not Installed)

Check if Python is installed:

```
python --version
```

If not installed, download and install **Python 3.8+** from:

🔗 <https://www.python.org/downloads/> (<https://www.python.org/downloads/>)

◆ Step 2: Create a Virtual Environment (Recommended)

A **virtual environment** helps manage dependencies without conflicts.

◆ For Windows (Command Prompt or PowerShell):

```
python -m venv nlp_env  
nlp_env\Scripts\activate
```

◆ For macOS/Linux (Terminal):

```
python3 -m venv nlp_env  
source nlp_env/bin/activate
```

🔗 Prerequisites for Learning NLP with NLTK 🚀

Before diving into **NLP with NLTK**, you should have some basic knowledge in the following areas:

● 1. Python Basics (Mandatory)

- ✓ Variables, Data Types (Strings, Lists, Dictionaries)
- ✓ Loops (`for` , `while`), Conditionals (`if-else`)
- ✓ Functions (`def` and `lambda` functions)
- ✓ File Handling (`open()` , `read()` , `write()`)

🔗 Recommended Learning:

🔗 [Python for Beginners \(W3Schools\)](https://www.w3schools.com/python/) (<https://www.w3schools.com/python/>)

● 2. Python Libraries for NLP

- ✓ **NLTK** – Core library for NLP tasks
- ✓ **NumPy** – Handling numerical operations
- ✓ **Pandas** – Data handling and processing

- ✓ **Matplotlib & Seaborn** – Data visualization
- ✓ **Scikit-learn** – Machine learning for text classification
- ✓ **Gensim** – Word embeddings and topic modeling

✦ **Installation:**

```
pip install nltk numpy pandas matplotlib seaborn scikit-learn gensim spacy
```

● 3. Basic Understanding of Machine Learning (Optional but Helpful)

- ✓ Concept of **training and testing data**
- ✓ Difference between **Supervised vs. Unsupervised Learning**
- ✓ Basic understanding of **Naïve Bayes Classifier** (used in text classification)

✦ **Recommended Learning:**

👉 [Machine Learning Basics \(Google\)](https://developers.google.com/machine-learning/crash-course) (<https://developers.google.com/machine-learning/crash-course>).

● 4. Some Knowledge of Linguistics (Helpful for NLP)

- ✓ What are **tokens, stopwords, and stemming**?

📌 Important Libraries for NLP in Python 🚀

To work efficiently in **Natural Language Processing (NLP)**, you need to use the right libraries. Below are the most important NLP libraries categorized by their functionalities.

◆ 1. Text Processing & Preprocessing Libraries

These libraries help in **tokenization, stopword removal, stemming, and lemmatization**.

✓ NLTK (Natural Language Toolkit)

- Best for beginners in NLP
- Provides tokenization, POS tagging, Named Entity Recognition (NER)
- **Installation:** `pip install nltk`

✓ spaCy

- Faster & more efficient than NLTK
- Supports dependency parsing & NER
- **Installation:**

```
pip install spacy
python -m spacy download en_core_web_sm # English Language model
```

✓ TextBlob

- Simple API for sentiment analysis, translation, and text classification
- **Installation:** `pip install textblob`
- **Example:**

```
from textblob import TextBlob
text = TextBlob("NLTK is a powerful tool!")
print(text.sentiment)
```

◆ 2. Machine Learning & Deep Learning for NLP

These libraries help with **text classification, embeddings, and deep learning models**.

✓ Scikit-Learn

- Useful for **text classification (Naïve Bayes, SVM, etc.)**
- Works well with NLTK for traditional ML-based NLP
- **Installation:** `pip install scikit-learn`

✓ TensorFlow & Keras

- Best for deep learning-based NLP (LSTMs, Transformers)
- Used for chatbot development, speech recognition
- **Installation:** `pip install tensorflow keras`

✓ PyTorch

- Used for advanced deep learning models (e.g., Transformers, BERT)
- **Installation:** `pip install torch`

✓ Hugging Face Transformers

- Provides **pre-trained models** like BERT, GPT, T5 for NLP tasks

- **Installation:** `pip install transformers`
- **Example:**

```
from transformers import pipeline
sentiment = pipeline("sentiment-analysis")
print(sentiment("I love NLP!"))
```

◆ 3. Word Embeddings & Semantic Analysis

These libraries help with **word representations** and understanding **semantic meaning**.

✓ Gensim

- Used for **word embeddings (Word2Vec, FastText, LDA)**
- **Installation:** `pip install gensim`

✓ FastText (by Facebook AI)

- Works well for **word embeddings** and **text classification**
- **Installation:** `pip install fasttext`

✓ SentenceTransformers

- Provides **BERT-based sentence embeddings** for similarity tasks
- **Installation:** `pip install sentence-transformers`

◆ 4. Text-to-Speech & Speech Recognition

These libraries help in **converting text to speech (TTS)** and **speech to text (STT)**.

✓ gTTS (Google Text-to-Speech)

- Converts text into speech
- **Installation:** `pip install gtts`

✓ SpeechRecognition

- Converts speech into text
- Works with Google Speech API, CMU Sphinx
- **Installation:** `pip install SpeechRecognition`

◆ 5. Other Useful NLP Libraries

✓ langdetect (Detects language from text)

- **Installation:** `pip install langdetect`

✓ polyglot (Multilingual NLP)

- Supports Named Entity Recognition (NER) in multiple languages
- **Installation:**

```
pip install polyglot
pip install pyicu
pip install morfessor
```

✓ DeepMultilingualNLP (Multilingual NLP tasks)

- Works well with **Hugging Face Transformers**
- **Installation:** `pip install deepmultilingualpunctuation`

★ Summary – Best NLP Libraries for Different Tasks

Task	Best Libraries
Text Processing	NLTK , spaCy , TextBlob
Machine Learning	Scikit-Learn , Hugging Face
Deep Learning	TensorFlow , PyTorch
Word Embeddings	Gensim , FastText , SentenceTransformers
Speech Recognition	SpeechRecognition , gTTS
Multilingual NLP	Polyglot , DeepMultilingualNLP

🚀 Let's start with NLTK (Natural Language Toolkit) for NLP (Natural Language

Processing) step by step.

Great! 🚀 Let's start with **NLTK (Natural Language Toolkit)** for **NLP (Natural Language Processing)** step by step.

◆ Step 1: Install & Set Up NLTK

✦ Install NLTK & Required Libraries

Run the following command in your terminal or Jupyter Notebook:

```
pip install nltk
```

For additional utilities:

```
pip install numpy pandas matplotlib seaborn scikit-learn
```

🚀 Next Steps

Text preprocessing and cleaning

We have covered:

- ✓ **Tokenization** (Word & Sentence Splitting)
- ✓ **Stopword Removal** (Filtering Common Words)
- ✓ **Stemming & Lemmatization** (Word Reduction)
- ✓ **POS Tagging** (Identifying Nouns, Verbs, etc.)
- ✓ **Named Entity Recognition (NER)** (Extracting Names, Places)
- ✓ **Sentiment Analysis** (Detecting Emotion in Text)

Tokenization

Basic Terminologies

1.corpus--> paragraphs

2.document--> sentences

3.vocabulary-->unique words

sentences Tokenization

nltk--->nltk.tokenize--->sent_tokenize

```
In [8]: import nltk

from nltk.tokenize import sent_tokenize

corpus="""Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI).
It enables computers to understand, interpret, and generate human language.
Popular NLP tasks include tokenization, part-of-speech tagging, named entity recognition (NER), and sentiment analysis.

NLTK (Natural Language Toolkit) is one of the most widely used libraries for NLP.
Other popular NLP libraries include SpaCy, Transformers (Hugging Face), and Stanford NLP.

Text preprocessing is an essential step in NLP. It includes tokenization, stopwords removal, stemming, and lemmatization.
Machine Learning and Deep Learning techniques are used to build intelligent NLP models.

Applications of NLP include chatbots, voice assistants, sentiment analysis, text summarization, and machine translation.
"""

sentence=sent_tokenize(corpus)

sent=[]
for i in sentence:
    sent.append(i) #just visualize only

# sent= " "

# for i in sentence:
#     sent+=i
```

◆ sent_tokenize()

✓ Advantages:

- Great for splitting paragraphs into meaningful sentences.
- Handles abbreviations and sentence-ending punctuation intelligently.

✖ Disadvantages:

- Can struggle with non-standard punctuation (e.g., ellipses, emojis).
- Language-dependent (works best in English unless trained for others).

In [3]: `len(sent)`

Out[3]: 9

In [4]: `sent`

Out[4]: ['Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI).',
'It enables computers to understand, interpret, and generate human language.',
'Popular NLP tasks include tokenization, part-of-speech tagging, named entity recognition (NER), and sentiment analysis.',
'NLTK (Natural Language Toolkit) is one of the most widely used libraries for NLP.',
'Other popular NLP libraries include SpaCy, Transformers (Hugging Face), and Stanford NLP.',
'Text preprocessing is an essential step in NLP.',
'It includes tokenization, stopword removal, stemming, and lemmatization.',
'Machine Learning and Deep Learning techniques are used to build intelligent NLP models.',
'Applications of NLP include chatbots, voice assistants, sentiment analysis, text summarization, and machine translation.']

In [5]: `sent`

Out[5]: ['Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI).',
'It enables computers to understand, interpret, and generate human language.',
'Popular NLP tasks include tokenization, part-of-speech tagging, named entity recognition (NER), and sentiment analysis.',
'NLTK (Natural Language Toolkit) is one of the most widely used libraries for NLP.',
'Other popular NLP libraries include SpaCy, Transformers (Hugging Face), and Stanford NLP.',
'Text preprocessing is an essential step in NLP.',
'It includes tokenization, stopword removal, stemming, and lemmatization.',
'Machine Learning and Deep Learning techniques are used to build intelligent NLP models.',
'Applications of NLP include chatbots, voice assistants, sentiment analysis, text summarization, and machine translation.']

In [6]: `# removing unwanted marks`

```
# import re
# ## removing unwanted marks (punctuation, numbers etc.)
# sentences=[]
# for sentence in sent:
#     clean_sentence=re.sub('[^a-z A-Z]', '', sentence)
#     clean_sentence+=" ".join(clean_sentence)

# print(clean_sentence)

import re

# Sample list of sentences

# Removing unwanted marks (punctuation, numbers, etc.)
cleaned_sentences = " "
for sentence in sent:
    cleaned_sentence = re.sub(r'[^a-zA-Z ]', '', sentence) # Keep only letters and spaces
    cleaned_sentences+=cleaned_sentence.strip() # Remove extra spaces

# Output the cleaned sentences

cleaned_sentences
```

Out[6]: ' Natural Language Processing NLP is a branch of Artificial Intelligence AIIt enables computers to understand interpret and generate human languagePopular NLP tasks include tokenization partofspeech tagging named entity recognition NER and sentime nt analysisNLTK Natural Language Toolkit is one of the most widely used libraries for NLPOther popular NLP libraries includ e SpaCy Transformers Hugging Face and Stanford NLPText preprocessing is an essential step in NLPIt includes tokenization st opword removal stemming and lemmatizationMachine Learning and Deep Learning techniques are used to build intelligent NLP mo delsApplications of NLP include chatbots voice assistants sentiment analysis text summarization and machine translation'

Here's a markdown version you can use for documentation, notebooks, or tutorials:

◆ Word Tokenization (Text Preprocessing Technique)

✓ Convert a sentence into individual words

Sample Sentence:

```
text = "There are different types of word tokenization."
```

1 Using word_tokenize from NLTK

```
import nltk
from nltk.tokenize import word_tokenize

# Download only the necessary tokenizer model
nltk.download('punkt') #  Recommended: Downloads only the Punkt tokenizer

# Tokenizing the sentence
tokens = word_tokenize(text)
print(tokens)
```

Note:

You can also download **all** NLTK resources using:

```
nltk.download("all")
```

But this will consume **over 2 GB** of space. 

Use it only if you need access to **all** corpora, models, and resources.

Output Example:

```
In [7]: import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

## we already have sentences

words=word_tokenize(cleaned_sentences)##corpus to sentences
stopword=[word for word in words if not word in set(stopwords.words('english'))]
stopword
```

```
Out[7]: ['Natural',
'Language',
'Processing',
'NLP',
'branch',
'Artificial',
'Intelligence',
'AIIt',
'enables',
'computers',
'understand',
'interpret',
'generate',
'human',
'languagePopular',
'NLP',
'tasks',
'include',
'tokenization',
'partofspeech',
'tagging',
'named',
'entity',
'recognition',
'NER',
'sentiment',
'analysisNLTK',
'Natural',
'Language',
'Toolkit',
'one',
'widely',
'used',
'libraries',
'NLPOther',
'popular',
'NLP',
'libraries',
'include',
'SpaCy',
'Transformers',
'Hugging',
'Face',
'Stanford',
'NLPText',
'preprocessing',
'essential',
'step',
'NLPIt',
'includes',
'tokenization',
'stopword',
'removal',
'stemming',
'lemmatizationMachine',
'Learning',
'Deep',
'Learning',
'techniques',
'used',
'build',
'intelligent',
'NLP',
'modelsApplications',
'NLP',
'include',
'chatbots',
'voice',
'assistants',
'sentiment',
'analysis',
'text',
'summarization',
'machine',
'translation']
```

◆ . word_tokenize()

✔ Advantages:

- Handles punctuation, contractions, and complex structures well.
- Built on the **Punkt** tokenizer + **Treebank rules**.
- Good default choice for many NLP tasks.

✗ Disadvantages:

- Slightly slower than simpler methods.
- May not work well on informal text (like tweets).

◆ 1. Regular Expression Tokenization

```
In [19]: from nltk.tokenize import RegexpTokenizer

Retokenize = RegexpTokenizer(r'\w+') # only words display

text="Hello! I am vigneshwaran, it's NLP END_END 100% " #remove ! and %

tokenize=Retokenize.tokenize(text) #output id List[]

for i in tokenize:
    print(i)
```

```
Hello
I
am
vigneshwaran
it
s
NLP
END_END
100
```

◆ RegexpTokenizer

✓ Advantages:

- Highly customizable via regex patterns.
- Can remove or keep only specific kinds of tokens (e.g., digits, words, symbols).

✗ Disadvantages:

- Requires regex knowledge.
- Not smart—won't handle linguistic nuances (e.g., "don't" vs. "do" + "n't").

◆ 3. WhitespaceTokenizer

Breaks on any whitespace—even if it's not a real word boundary. Ignores punctuation. ('occure.Ignore',)

```
In [29]: from nltk.tokenize import WhitespaceTokenizer

Whitespace=WhitespaceTokenizer()
Text1=" whitespace Tokenizer word split based on the space only occure.Ignore punctuation"
tokens=Whitespace.tokenize(Text1)

tokens
```

```
Out[29]: ['whitespace',
'Tokenizer',
'word',
'split',
'based',
'on',
'the',
'space',
'only',
'occure.Ignore',
'punctuation']
```

◆ WhitespaceTokenizer

✓ Advantages:

- Very fast and simple.
- No external models or rules needed.

✗ Disadvantages:

- Ignores punctuation.
- Breaks on any whitespace—even if it's not a real word boundary.
- Poor for real-world text where spacing is inconsistent.

◆ TreebankWordTokenizer

4 Using TreebankWordTokenizer from NLTK

The **Treebank Word Tokenizer** uses the tokenization conventions of the **Penn Treebank**. It handles contractions and punctuations more precisely than basic tokenizers.

✦ Features:

- Splits **contractions** (e.g., don't → do + n't)
- Separates **punctuation marks** as individual tokens
- Follows linguistic structure useful in **syntactic parsing**

```
In [43]: from nltk.tokenize import TreebankWordTokenizer

Tree = TreebankWordTokenizer()

text2="Don't like the Pizza "

Tree.tokenize(text2)
```

```
Out[43]: ['Do', "n't", 'like', 'the', 'Pizza']
```

◆ TreebankWordTokenizer

✔ Advantages:

- Follows Penn Treebank tokenization standards.
- Good for syntactic parsing tasks.
- Splits contractions (e.g., “don’t” → “do” + “n’t”).

✗ Disadvantages:

- Not as robust for social or informal text.
- More rule-based; may not adapt to new forms or misspellings

✗ Avoid if: You’re processing informal/social media text (use TweetTokenizer instead)

You don't need detailed syntactic features

📚 Related Task: TreebankTokenizer is commonly used in NLP tasks like:

POS Tagging

Dependency Parsing

Named Entity Recognition (NER)

◆ TweetTokenizer

Certainly, Vicky! Let's delve into the **TweetTokenizer** provided by NLTK, specifically designed for tokenizing content from social media platforms like Twitter.

◆ TweetTokenizer Overview

□ The **TweetTokenizer** is tailored to handle the unique challenges presented by tweets, such as □ □

- **User Mentions:** □ Recognizes and treats handles (e.g., @username) as distinct tokens □ □
- **Hashtags:** □ Identifies hashtags (e.g., #Topic) as individual tokens □ □
- **Emojis and Emoticons:** □ Preserves emojis and common emoticons, ensuring they are tokenized correctly □ □
- **URLs:** □ Can isolate URLs within the text □ □

◆ Key Features and Parameters

When initializing **TweetTokenizer** , you can customize its behavior with the following parameters:

- **preserve_case** (default: `True`) □ Determines whether to retain the original casing. Setting it to `False` converts all tokens to lowercase. □ □
- **reduce_len** (default: `False`) □ When set to `True` , it shortens elongated character sequences (e.g., "cooooool" becomes "coool". □ □
- **strip_handles** (default: `False`) □ If `True` , removes Twitter handles (e.g., @user) from the tokenized output. □ □

◆ Usage Example

Here's how you can utilize TweetTokenizer :

```
from nltk.tokenize import TweetTokenizer

# Initialize the tokenizer with desired settings
tweet_tokenizer = TweetTokenizer(preserve_case=False, reduce_len=True, strip_handles=True)

# Sample tweet
tweet = "@Vicky This is amaaazing! 🤩 #excited http://example.com"

# Tokenize the tweet
tokens = tweet_tokenizer.tokenize(tweet)

print(tokens)
```

Output:

```
[':', 'this', 'is', 'amaaazing', '!', '🤩', '#excited', 'http://example.com']
```

Explanation:

- **Handles Remove:** The @Vicky mention is removed due to strip_handles=True.
- **Lowercased Token:** All tokens are in lowercase because preserve_case=False.
- **Reduced Lengthen:** "amaaazing" is shortened to "amaaazing" (from "amaaaazing") because of reduce_len=True.
- **Emojis and Hashtags Preserve:** The emoji 🤩 and hashtag #excited are retained as individual tokens.

◆ When to Use TweetTokenizer

Opt for TweetTokenizer when dealing with text from social media platforms, especially Twitter, where

- Special elements like mentions, hashtags, and emojis are prealen.
- Standard tokenizers might not handle the informal and varied structures effectively.

For more detailed information, you can refer to the [NLTK documentation on TweetTokenizer \(https://www.nltk.org/api/nltk.tokenize.casual.html\)](https://www.nltk.org/api/nltk.tokenize.casual.html).

```
In [39]: from nltk.tokenize import TweetTokenizer
```

```
In [41]: Tweet_tokenizer=TweetTokenizer(preserve_case=True,reduce_len=True)

tweet = "@Vicky This is amaaazing! 🤩 #excited http://example.com"

token=Tweet_tokenizer.tokenize(tweet)

token
```

```
Out[41]: ['@Vicky',
'This',
'is',
'amaaazing',
'!',
'🤩',
'#excited',
'http://example.com']
```

◆ 6. TweetTokenizer

✓ Advantages:

- Designed for tweets and social media content.
- Keeps hashtags, mentions, emojis, and URLs as separate tokens.
- Handles informal writing better than other tokenizers.

✗ Disadvantages:

- Not ideal for formal text (e.g., news articles).
- May keep too much noise (e.g., emojis or junk tokens).

⚡ Summary Table

Tokenizer	Best For	Pros	Cons
word_tokenize	General text (news, books)	Accurate, handles grammar well	Slower, less social text support
sent_tokenize	Paragraphs, documents	Sentence-aware	Misfires on unusual punctuation
RegexpTokenizer	Custom patterns	Fully customizable	Needs regex knowledge

Tokenizer	Best For	Pros	Cons
WhitespaceTokenizer	Clean, preprocessed text	Super fast	Ignores punctuation
TreebankWordTokenizer	Parsing, structured tasks	Standardized, splits contractions	Not robust for informal text
TweetTokenizer	Tweets, chats, social media	Emoji/hashtag-friendly	May be noisy

◆ Stemming

Stemming in NLP (Natural Language Processing) is the process of reducing a word to its base or root form. The idea is to remove suffixes or prefixes to get to the core part of the word, called the "stem."

🔍 Why Use Stemming? In NLP, you want to treat different forms of a word (like run, running, ran) as the same word so that:

1. Search engines can return better results
2. Text classification becomes more accurate
3. Topic modeling and sentiment analysis are more consistent

◆ 1.1. Porter Stemmer (Balanced)

- ✅ **Less aggressive**, produces more **readable stems**
- ✅ Performs well in **general NLP tasks**
- ❌ May still output **some incorrect or unnatural stems** (e.g., "relational" → "relat")

★ Best For:

- Search engines
- Text normalization
- Topic modeling
- Preprocessing for text classification

nltk.stem module --> PorterStemmer(class)

```
In [6]: from nltk.stem import PorterStemmer #class

#create object
stemmer = PorterStemmer()

#example words
words = ['playing', 'play', 'Played', 'relational', 'running']

#stemming processing
stems = [stemmer.stem(word) for word in words ]

stems
```

Out[6]: ['play', 'play', 'play', 'relat', 'run']

2. Lancaster Stemmer (Aggressive)

--- ✅ Very aggressive, produces shorter stems --- ✅ Faster than Porter but sacrifices accuracy --- ❌ Over-stems words, making them hard to interpret

👉 Best for: When you need high compression of words, but accuracy isn't critical.

✅ "High compression of words" means: The stemmer reduces many similar words to a very short or basic form.

For example:

"connection", "connected", "connecting" → all become "connect" or even just "connect" → "connect" or "connect" → "connect" (in aggressive stemmers, it might even be "conn").

⚠️ "Accuracy isn't critical" means: It's okay if the output isn't perfect or grammatically correct.

Example: "maximum" becoming "maxim" is not ideal, but acceptable in some cases.

You're trading accuracy for performance or simplicity.

```
In [22]: from nltk.stem import LancasterStemmer

#create the object
lan=LancasterStemmer()

#words
words = ["connection", "connected", "connecting", "maximum", "running", "better"]

#just understanding prepose
print("word      | lancaSTERstemmer")
print('-----')

for word in words:
    lan=lancaSTERstemmer.stem(word)
    print(f'{word:<10} | {lan:<100}')
```

```
word      | lancaSTERstemmer
-----
connection | connect
connected  | connect
connecting | connect
maximum    | maxim
running    | run
better     | bet
```

3.RegexpStemmer class

🌸 So When to Use It? ☒ You know your data (e.g., custom logs, forms, special language)

☒ You want lightweight, fast, and specific stemming

✗ Not for general-purpose NLP (use Porter or Snowball instead)

⚠ Limitations: No intelligence — it just chops text based on pattern

Not suitable for complex language cases

You must design the regex carefully

```
In [38]: from nltk.stem import RegexpStemmer

#remove common verb suffixes
stemmer=RegexpStemmer("(ing|ed|s)$")

words=["playing", "ended", "indians"]

print("word | RegexpStemmer ")
print("-----")

# Reg=[stemmer.stem(word) for word in words]

for word in words:
    stemmed = stemmer.stem(word)
    print(f'{word:<10} | {stemmed:<10}')
```

```
word | RegexpStemmer
-----
playing | play
ended   | end
indians | indian
```

❄ Snowball Stemmer (also called the English Stemmer)

1.It's a more advanced version of the Porter Stemmer.

2.Supports multiple languages.

3.Gives better and more consistent results than basic stemmers

```
In [40]: #SnowballStemmer.Languages
# ['arabic', 'danish', 'dutch', 'english', 'finnish', 'french',
# 'german', 'hungarian', 'italian', 'norwegian', 'porter',
# 'portuguese', 'romanian', 'russian', 'spanish', 'swedish', 'turkish']
```

```
In [18]: from nltk.stem import SnowballStemmer

Snowball=SnowballStemmer("english")

words = [
    "playing", "played", "player", "plays",      # verb forms
    "running", "runner", "ran",                  # verb variations
    "happiness", "happy", "happily",             # noun/adjective/adverb
    "connected", "connection", "connecting",      # related words
    "studies", "studied", "studying", "student",  # education-related
    "flies", "flying", "fly",                    # plural vs singular
    "better", "best", "good",                    # irregular comparison
    "maximum", "minimum", "optimum",             # test over-stemming
    "national", "nation", "nationality",         # root relationship
    "agreement", "agreeing", "agreed",           # common NLP stem group
]

print('word | Snowballstemmer')
print("-----")

for word in words:
    snow=Snowball.stem(word)
    print(f'{word:<10} | {snow:<10}')
```

word | Snowballstemmer

```
-----
playing | play
played  | play
player  | player
plays   | play
running | run
runner  | runner
ran     | ran
happiness | happi
happy   | happi
happily | happili
connected | connect
connection | connect
connecting | connect
studies | studi
studied | studi
studying | studi
student | student
flies   | fli
flying  | fli
fly     | fli
better  | better
best    | best
good    | good
maximum | maximum
minimum | minimum
optimum | optimum
national | nation
nation  | nation
nationality | nation
agreement | agreement
agreeing | agree
agreed  | agree
```

Lemmatization in a clear and simple way — it's a core concept in **NLP (Natural Language Processing)**.

What is Lemmatization?

Lemmatization is the process of converting a word to its **base or dictionary form**, called the **lemma**.

✔ It takes **context and grammar** into account.

Examples:

Original Word	Lemma	Part of Speech
running	run	verb
better	good	adjective
studies	study	verb
children	child	noun

Why Use Lemmatization?

It helps in:

- Reducing **vocabulary size** in ML models
- Grouping similar words (e.g., "run", "running", "ran")

- Improving **text search, classification, chatbot responses**, etc.

Example:

Without lemmatization, "running" and "ran" are treated as different.
With lemmatization, both become **"run"** – easier to analyze.

 Difference: Lemmatization vs Stemming

Feature	Lemmatization	Stemming
Meaning-aware	✔ Yes	✘ No
Grammar-aware	✔ Yes	✘ No
Accurate	✔ More accurate	✘ Rough cut
Example ("better")	→ "good" (correct)	→ "bett" (wrong)
Example ("studies")	→ "study"	→ "studi"

 Python Code Example using spaCy

```
import spacy

nlp = spacy.load("en_core_web_sm")

doc = nlp("The children were running faster than their parents")

for token in doc:
    print(f"{token.text} -> {token.lemma_}")
```

Output:

```
The -> the
children -> child
were -> be
running -> run
faster -> fast
than -> than
their -> their
parents -> parent
```

 Libraries That Support Lemmatization

Library	Method
spaCy	token.lemma_
NLTK	WordNetLemmatizer
TextBlob	word.lemmatize()
Stanza	word.lemma

 Pro Tip:

In LLM apps (like Chatbots or search engines), **lemmatization** helps improve understanding of:

"I was running" vs "He runs fast" → both boil down to the verb **run**

If you're building an AI app, I can help you integrate lemmatization into:

- Text preprocessing pipelines
- Search engines
- Chatbots
- Sentiment analysis

```
In [23]: #In nltk
# import nltk
# nltk.download('wordnet')
# nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
words = [
    "playing", "played", "player", "plays",      # verb forms
    "running", "runner", "ran",                 # verb variations
    "happiness", "happy", "happily",             # noun/adjective/adverb
    "connected", "connection", "connecting",     # related words
    "studies", "studied", "studying", "student", # education-related
    "flies", "flying", "fly",                   # plural vs singular
    "better", "best", "good",                   # irregular comparison
    "maximum", "minimum", "optimum",            # test over-stemming
    "national", "nation", "nationality",         # root relationship
    "agreement", "agreeing", "agreed",          # common NLP stem group
]

print(" words | lemmatized | stemming")
print("-----")
#use object:
for i in words:
    lem=lemmatizer.lemmatize(i,pos="v")
    stem=snow=Snowball.stem(i)
#    print(f'{words:<10} | {lem:<10}')
    print(f'{i:<10} | {lem:<10} | {stem:<10} ')

```

```
words | lemmatized | stemming
-----
playing | play | play
played | play | play
player | player | player
plays | play | play
running | run | run
runner | runner | runner
ran | run | ran
happiness | happiness | happi
happy | happy | happi
happily | happily | happili
connected | connect | connect
connection | connection | connect
connecting | connect | connect
studies | study | studi
studied | study | studi
studying | study | studi
student | student | student
flies | fly | fli
flying | fly | fli
fly | fly | fli
better | better | better
best | best | best
good | good | good
maximum | maximum | maximum
minimum | minimum | minimum
optimum | optimum | optimum
national | national | nation
nation | nation | nation
nationality | nationality | nation
agreement | agreement | agreement
agreeing | agree | agree
agreed | agree | agree

```

Stopwords



What are Stopwords?

Stopwords are the **common words** in a language that:

- Occur frequently (like "is", "the", "and", "a", "an", "to", etc.)
- Don't carry much **semantic meaning**
- Are often **removed** during text preprocessing



Example Sentence:

"The sun is shining brightly in the sky"



Stopwords Removed:

"sun shining brightly sky"



The words: the, is, in, the are **stopwords**

? Why Remove Stopwords?

- ✓ To **reduce noise** in data
- ✓ To **improve performance** of ML/NLP models
- ✓ To focus on **important keywords**

Common Libraries with Stopword Support

Library	How to Use
NLTK	<code>from nltk.corpus import stopwords</code>
spaCy	<code>nlp.Defaults.stop_words</code>
sklearn	<code>CountVectorizer(stop_words='english')</code>

Example with NLTK:

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

nltk.download('punkt')
nltk.download('stopwords')

text = "This is a simple sentence with some common words"
words = word_tokenize(text)

filtered = [w for w in words if w.lower() not in stopwords.words('english')]

print(filtered)
```

✓ Output:

```
['simple', 'sentence', 'common', 'words']
```

Important Notes:

- Not all stopwords are useless — **depends on your task**.
- In **sentiment analysis**, words like **"not"** or **"never"** can change meaning → don't remove blindly!

Use Cases of Stopword Removal:

Task	Should Remove Stopwords?
Text classification	✓ Usually yes
Search engines	✓ Yes
Sentiment analysis	⚠ Be careful
Summarization/Chatbots	✗ Sometimes better to keep

```
In [97]: #default stopword supported languages
stopwords.words("english")
# stopwords.words("arabic")
```

```
Out[97]: ['a',
'about',
'above',
'after',
'again',
'against',
'ain',
'all',
'am',
'an',
'and',
'any',
'are',
'aren',
'aren't',
'as',
'at',
'be',
'because',
'-----']
```

```
In [5]: #while using
# nltk.download('punkt')
# nltk.download('stopwords')

corpus="""
-- Agentic AI Corpus --

-- Definition of Agentic AI:
Agentic AI refers to artificial intelligence systems that can initiate actions, make decisions, and pursue goals independently, often simulating human-like autonomy. These agents are capable of planning, reasoning, and interacting with environments or users without continuous external instruction.

-- Key Characteristics:
Agentic AI systems are proactive, goal-oriented, context-aware, and adaptive. Unlike passive models, agentic systems make decisions based on real-time data and feedback loops. They may integrate with tools, APIs, or external knowledge sources to achieve tasks.

-- Examples of Agentic AI:
Examples include AutoGPT, BabyAGI, OpenAI's GPT agents, and Hugging Face Transformers in agentic frameworks. These models chain prompts, monitor outcomes, and make iterative decisions. For instance, an agent could perform research, write code, and evaluate results autonomously.

-- Applications:
Agentic AI is applied in domains like autonomous customer service bots, personal AI assistants, smart financial advisors, coding copilots, and scientific research tools. They are also used in multi-agent simulations for planning and optimization.

-- Challenges:
Challenges include alignment with user intent, safety, hallucination control, and ethical decision-making. There's also a need for transparent reasoning, sandboxed tool usage, and responsible autonomy to avoid unintended consequences.

-- Comparison: Agentic AI vs Traditional AI:
Traditional AI responds to input with fixed outputs. Agentic AI can evaluate multiple actions, explore options, and adapt strategies dynamically. It shifts from 'input-output' to 'goal-strategy-feedback'.

-- Future Trends:
Future directions include self-repairing agents, multi-modal agent orchestration, personal AGI tutors, decentralized agent ecosystems, and collaborative workflows with humans in the loop.

"""
```

```
In [6]: from nltk.tokenize import sent_tokenize

sentence=sent_tokenize(corpus)
```

```
In [7]: sentence
```

```
Out[7]: ['\n-- Agentic AI Corpus --\n\n-- Definition of Agentic AI:\nAgentic AI refers to artificial intelligence systems that can initiate actions, make decisions, and pursue goals independently, \noften simulating human-like autonomy.',
'These agents are capable of planning, reasoning, and interacting with environments or users \nwithout continuous external instruction.',
'-- Key Characteristics:\nAgentic AI systems are proactive, goal-oriented, context-aware, and adaptive.',
'Unlike passive models, agentic systems make \ndecisions based on real-time data and feedback loops.',
'They may integrate with tools, APIs, or external knowledge sources to \nachieve tasks.',
'-- Examples of Agentic AI:\nExamples include AutoGPT, BabyAGI, OpenAI's GPT agents, and Hugging Face Transformers in agentic frameworks.',
'These models \nchain prompts, monitor outcomes, and make iterative decisions.',
'For instance, an agent could perform research, write code, and \nevaluate results autonomously.',
'-- Applications:\nAgentic AI is applied in domains like autonomous customer service bots, personal AI assistants, smart financial advisors, \ncoding copilots, and scientific research tools.',
'They are also used in multi-agent simulations for planning and optimization.',
'-- Challenges:\nChallenges include alignment with user intent, safety, hallucination control, and ethical decision-making.',
'There's also a \nneed for transparent reasoning, sandboxed tool usage, and responsible autonomy to avoid unintended consequences.',
'-- Comparison: Agentic AI vs Traditional AI:\nTraditional AI responds to input with fixed outputs.',
'Agentic AI can evaluate multiple actions, explore options, and adapt \nstrategies dynamically.',
'It shifts from 'input-output' to 'goal-strategy-feedback'.',
'-- Future Trends:\nFuture directions include self-repairing agents, multi-modal agent orchestration, personal AGI tutors, decentralized agent \necosystems, and collaborative workflows with humans in the loop.']
```

```
In [46]: import re

# cleaned_sentences = []

# for sent in sentences:
#     cleaned = re.sub('[^a-zA-Z]', ' ', sent)
#     cleaned_sentences.append(cleaned)

# cleaned_sentences = []

for sent in sentence:
    cleaned = re.sub('[^a-zA-Z]', ' ', sent) # Remove non-Letter characters
    cleaned = re.sub(r'\s+', ' ', cleaned) # Normalize spaces
    cleaned_sentences.append(cleaned) # Add cleaned text to list

print(cleaned_sentences)
```

['Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions make decisions and pursue goals independently often simulating human like autonomy', 'These agents are capable of planning reasoning and interacting with environments or users without continuous external instruction', 'Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive', 'Unlike passive models agentic systems make decisions based on real time data and feedback loops', 'They may integrate with tools APIs or external knowledge sources to achieve tasks', 'Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic frameworks', 'These models chain prompts monitor outcomes and make iterative decisions', 'For instance an agent could perform research write code and evaluate results autonomously', 'Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financial advisors coding copilots and scientific research tools', 'They are also used in multi agent simulations for planning and optimization', 'Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making', 'There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequences', 'Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs', 'Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically', 'It shifts from input output to goal strategy feedback', 'Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentralized agent ecosystems and collaborative workflows with humans in the loop', ' Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions, make decisions, and pursue goals independently, often simulating human like autonomy ', 'These agents are capable of planning, reasoning, and interacting with environments or users without continuous external instruction ', ' Key Characteristics Agentic AI systems are proactive, goal oriented, context aware, and adaptive ', 'Unlike passive models, agentic systems make decisions based on real time data and feedback loops ', 'They may integrate with tools, APIs, or external knowledge sources to achieve tasks ', ' Examples of Agentic AI Examples include AutoGPT, BabyAGI, OpenAI s GPT agents, and Hugging Face Transformers in agentic frameworks ', 'These models chain prompts, monitor outcomes, and make iterative decisions ', 'For instance, an agent could perform research, write code, and evaluate results autonomously ', ' Applications Agentic AI is applied in domains like autonomous customer service bots, personal AI assistants, smart financial advisors, coding copilots, and scientific research tools ', 'They are also used in multi agent simulations for planning and optimization ', ' Challenges Challenges include alignment with user intent, safety, hallucination control, and ethical decision making ', 'There s also a need for transparent reasoning, sandboxed tool usage, and responsible autonomy to avoid unintended consequences ', ' Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs ', 'Agentic AI can evaluate multiple actions, explore options, and adapt strategies dynamically ', 'It shifts from input output to goal strategy feedback ', ' Future Trends Future directions include self repairing agents, multi modal agent orchestration, personal AGI tutors, decentralized agent ecosystems, and collaborative workflows with humans in the loop ']

In [12]: cleaned_sentences

```
Out[12]: ['Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions
make decisions and pursue goals independently often simulating human like autonomy',
'These agents are capable of planning reasoning and interacting with environments or users without continuous external ins
truction',
'Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive',
'Unlike passive models agentic systems make decisions based on real time data and feedback loops',
'They may integrate with tools APIs or external knowledge sources to achieve tasks',
'Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic fram
eworks',
'These models chain prompts monitor outcomes and make iterative decisions',
'For instance an agent could perform research write code and evaluate results autonomously',
'Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financia
l advisors coding copilots and scientific research tools',
'They are also used in multi agent simulations for planning and optimization',
'Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making',
'There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequen
ces',
'Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs',
'Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically',
'It shifts from input output to goal strategy feedback',
'Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentr
alized agent ecosystems and collaborative workflows with humans in the loop',
'Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions
make decisions and pursue goals independently often simulating human like autonomy',
'These agents are capable of planning reasoning and interacting with environments or users without continuous external ins
truction',
'Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive',
'Unlike passive models agentic systems make decisions based on real time data and feedback loops',
'They may integrate with tools APIs or external knowledge sources to achieve tasks',
'Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic fram
eworks',
'These models chain prompts monitor outcomes and make iterative decisions',
'For instance an agent could perform research write code and evaluate results autonomously',
'Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financia
l advisors coding copilots and scientific research tools',
'They are also used in multi agent simulations for planning and optimization',
'Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making',
'There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequen
ces',
'Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs',
'Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically',
'It shifts from input output to goal strategy feedback',
'Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentr
alized agent ecosystems and collaborative workflows with humans in the loop',
'Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions
make decisions and pursue goals independently often simulating human like autonomy',
'These agents are capable of planning reasoning and interacting with environments or users without continuous external ins
truction',
'Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive',
'Unlike passive models agentic systems make decisions based on real time data and feedback loops',
'They may integrate with tools APIs or external knowledge sources to achieve tasks',
'Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic fram
eworks',
'These models chain prompts monitor outcomes and make iterative decisions',
'For instance an agent could perform research write code and evaluate results autonomously',
'Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart finan
cial advisors coding copilots and scientific research tools',
'They are also used in multi agent simulations for planning and optimization',
'Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making',
'There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended conseq
uences',
'Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs',
'Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically',
'It shifts from input output to goal strategy feedback',
'Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors dec
entralized agent ecosystems and collaborative workflows with humans in the loop']
```



```

In [58]: import re
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
import nltk

# Download if not already done
nltk.download('punkt')
nltk.download('stopwords')

corpus="""
-- Agentic AI Corpus --

-- Definition of Agentic AI:
Agentic AI refers to artificial intelligence systems that can initiate actions, make decisions, and pursue goals independently, often simulating human-like autonomy. These agents are capable of planning, reasoning, and interacting with environments or users without continuous external instruction.

-- Key Characteristics:
Agentic AI systems are proactive, goal-oriented, context-aware, and adaptive. Unlike passive models, agentic systems make decisions based on real-time data and feedback loops. They may integrate with tools, APIs, or external knowledge sources to achieve tasks.

-- Examples of Agentic AI:
Examples include AutoGPT, BabyAGI, OpenAI's GPT agents, and Hugging Face Transformers in agentic frameworks. These models chain prompts, monitor outcomes, and make iterative decisions. For instance, an agent could perform research, write code, and evaluate results autonomously.

-- Applications:
Agentic AI is applied in domains like autonomous customer service bots, personal AI assistants, smart financial advisors, coding copilots, and scientific research tools. They are also used in multi-agent simulations for planning and optimization.

-- Challenges:
Challenges include alignment with user intent, safety, hallucination control, and ethical decision-making. There's also a need for transparent reasoning, sandboxed tool usage, and responsible autonomy to avoid unintended consequences.

-- Comparison: Agentic AI vs Traditional AI:
Traditional AI responds to input with fixed outputs. Agentic AI can evaluate multiple actions, explore options, and adapt strategies dynamically. It shifts from 'input-output' to 'goal-strategy-feedback'.

-- Future Trends:
Future directions include self-repairing agents, multi-modal agent orchestration, personal AGI tutors, decentralized agent ecosystems, and collaborative workflows with humans in the loop.

"""

# 1. Sentence Tokenization
sentences = sent_tokenize(corpus)

# 2. Clean and store sentences

#method1
cleaned_sentences = []

for sent in sentences:
    cleaned = re.sub('[^a-zA-Z]', ' ', sent)# Remove non-Letter characters
    # "This is a test"
    cleaned = re.sub(r'\s+', ' ', cleaned)# Normalize spaces
    # "This is a test"
    cleaned_sentences.append(cleaned.strip()) # Add cleaned text to list
    # " Agentic AI is powerful " → "Agentic AI is powerful"

#method2
cleaned_sentences1 = " "

for sent in sentences:
    cleaned = re.sub('[^a-zA-Z]', ' ', sent)# Remove non-Letter characters
    # "This is a test"
    cleaned = re.sub(r'\s+', ' ', cleaned)# Normalize spaces
    # "This is a test"
    # cleaned_sentences.append(cleaned.strip()) # Add cleaned text to list
    # " Agentic AI is powerful " → "Agentic AI is powerful"
    cleaned_sentences1+=cleaned.strip() + " " # " " separate the sentence

print(cleaned_sentences1)

#method 3:

cleaned_sentences2 = []

for sent in sentences:
    cleaned = re.sub('[^a-zA-Z]', ' ', sent)# Remove non-Letter characters
    cleaned = re.sub(r'\s+', ' ', cleaned)# Normalize spaces
    cleaned_sentences2.append(cleaned.strip())

#string join formate

```

```
doc=" ".join(cleaned_sentences2)

print(doc)
```

Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions make decisions and pursue goals independently often simulating human like autonomy. These agents are capable of planning reasoning and interacting with environments or users without continuous external instruction. Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive. Unlike passive models agentic systems make decisions based on real time data and feedback loops. They may integrate with tools APIs or external knowledge sources to achieve tasks. Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic frameworks. These models chain prompts monitor outcomes and make iterative decisions. For instance an agent could perform research write code and evaluate results autonomously. Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financial advisors coding copilots and scientific research tools. They are also used in multi agent simulations for planning and optimization. Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making. There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequences. Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs. Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically. It shifts from input output to goal strategy feedback . Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentralized agent ecosystems and collaborative workflows with humans in the loop.

Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions make decisions and pursue goals independently often simulating human like autonomy. These agents are capable of planning reasoning and interacting with environments or users without continuous external instruction. Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive. Unlike passive models agentic systems make decisions based on real time data and feedback loops. They may integrate with tools APIs or external knowledge sources to achieve tasks. Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic frameworks. These models chain prompts monitor outcomes and make iterative decisions. For instance an agent could perform research write code and evaluate results autonomously. Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financial advisors coding copilots and scientific research tools. They are also used in multi agent simulations for planning and optimization. Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making. There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequences. Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs. Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically. It shifts from input output to goal strategy feedback . Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentralized agent ecosystems and collaborative workflows with humans in the loop.

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

In [59]: `len(cleaned_sentences)`

Out[59]: 16

```
In [60]: #using method 3
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.stem import WordNetLemmatizer
```

```
#corpus->doc

doc1=sent_tokenize(doc)

doc1
```

Out[60]: ['Agentic AI Corpus Definition of Agentic AI Agentic AI refers to artificial intelligence systems that can initiate actions make decisions and pursue goals independently often simulating human like autonomy.',
'These agents are capable of planning reasoning and interacting with environments or users without continuous external instruction.',
'Key Characteristics Agentic AI systems are proactive goal oriented context aware and adaptive.',
'Unlike passive models agentic systems make decisions based on real time data and feedback loops.',
'They may integrate with tools APIs or external knowledge sources to achieve tasks.',
'Examples of Agentic AI Examples include AutoGPT BabyAGI OpenAI s GPT agents and Hugging Face Transformers in agentic frameworks.',
'These models chain prompts monitor outcomes and make iterative decisions.',
'For instance an agent could perform research write code and evaluate results autonomously.',
'Applications Agentic AI is applied in domains like autonomous customer service bots personal AI assistants smart financial advisors coding copilots and scientific research tools.',
'They are also used in multi agent simulations for planning and optimization.',
'Challenges Challenges include alignment with user intent safety hallucination control and ethical decision making.',
'There s also a need for transparent reasoning sandboxed tool usage and responsible autonomy to avoid unintended consequences.',
'Comparison Agentic AI vs Traditional AI Traditional AI responds to input with fixed outputs.',
'Agentic AI can evaluate multiple actions explore options and adapt strategies dynamically.',
'It shifts from input output to goal strategy feedback .',
'Future Trends Future directions include self repairing agents multi modal agent orchestration personal AGI tutors decentralized agent ecosystems and collaborative workflows with humans in the loop.']

In [56]: `len(doc1)`

Out[56]: 16

```
In [94]: # for j in range(0, len(doc1)):
stemming = SnowballStemmer("english")
lemma = WordNetLemmatizer()
for review in doc1:
    review = review.lower()
    review = word_tokenize(review)
    review2 = [stemming.stem(words) for words in review if not words in set(stopwords.words("english"))]
    review1 = [lemma.lemmatize(word) for word in review if not word in set(stopwords.words("english"))]
    #string
    review = " ".join(review1)
    review2 = " ".join(review2)
    print(review, "\n\n")
    print(review3)
```

agentic ai corpus definition agentic ai agentic ai refers artificial intelligence system initiate action make decision pursue goal independently often simulating human like autonomy .

agent ai corpus definition agent ai agent ai refer artificial intelligence system initiate action make decision pursue goal independently often simulate human like autonomy .

agent capable planning reasoning interacting environment user without continuous external instruction .

agent capable plan reason interact environment user without continuous external instruction .

key characteristic agentic ai system proactive goal oriented context aware adaptive .

key characteristic agent ai system proactive goal oriented context aware adaptive .

unlike passive model agentic system make decision based real time data feedback loop .

unlike passive model agent system make decision based real time data feedback loop .

may integrate tool apis external knowledge source achieve task .

may integrate tool apis external knowledge source achieve task .

example agentic ai example include autogpt babyagi openai gpt agent hugging face transformer agentic framework .

example agent ai example include autogpt babyagi openai gpt agent hugging face transformer agent framework .

model chain prompt monitor outcome make iterative decision .

model chain prompt monitor outcome make iterative decision .

instance agent could perform research write code evaluate result autonomously .

instance agent could perform research write code evaluate result autonomously .

application agentic ai applied domain like autonomous customer service bot personal ai assistant smart financial advisor coding copilot scientific research tool .

application agent ai applied domain like autonomous customer service bot personal ai assistant smart financial advisor coding copilot scientific research tool .

also used multi agent simulation planning optimization .

also use multi agent simulation plan optimization .

challenge challenge include alignment user intent safety hallucination control ethical decision making .

challenge challenge include align user intent safety hallucination control ethical decision making .

also need transparent reasoning sandboxed tool usage responsible autonomy avoid unintended consequences .

also need transparent reasoning sandbox tool usage responsible autonomy avoid unintended consequences .

comparison agentic ai v traditional ai traditional ai responds input fixed output .

comparison agent ai vs traditional ai traditional ai responds input fixed output .

agentic ai evaluate multiple action explore option adapt strategy dynamically .

agent ai evaluate multiple action explore option adapt strategy dynamically .

shift input output goal strategy feedback .

shift input output goal strategy feedback .

future trend future direction include self repairing agent multi modal agent orchestration personal agent tutor decentralized agent ecosystem collaborative workflow human loop .

future trend future direction include self repair agent multi modal agent orchestration personal agent tutor decentralized agent ecosystem collaborative workflow human loop .

Parts of speech tag

Nouns

- NN – Noun, singular or mass
 - NNS – Noun, plural
 - NNP – Proper noun, singular
 - NNPS – Proper noun, plural
-

Pronouns

- PRP – Personal pronoun (he , she , they)
 - PRP\$ – Possessive pronoun (his , her , their)
 - WP – Wh-pronoun (who , what)
 - WP\$ – Possessive wh-pronoun (whose)
-

Determiners & Related

- DT – Determiner (the , a , an)
 - PDT – Predeterminer (all , both)
 - WDT – Wh-determiner (which , that)
 - EX – Existential “there”
-

Verbs

- VB – Base form (run)
 - VBD – Past tense (ran)
 - VBG – Gerund/Present participle (running)
 - VBN – Past participle (run , eaten)
 - VBP – Non-3rd person singular present (run)
 - VBZ – 3rd person singular present (runs)
 - MD – Modal (can , will)
-

Adjectives

- JJ – Adjective
 - JJR – Comparative adjective (better)
 - JJS – Superlative adjective (best)
-

Adverbs

- RB – Adverb
 - RBR – Comparative adverb (faster)
 - RBS – Superlative adverb (fastest)
 - WRB – Wh-adverb (where , when , how)
-

Conjunctions & Prepositions

- CC – Coordinating conjunction (and , but)
 - IN – Preposition or subordinating conjunction (in , because)
 - TO – “to”
-

Other Useful Tags

- RP – Particle (up in “give up”)
 - FW – Foreign word
 - SYM – Symbol (\$, % , +)
 - UH – Interjection (uh , wow)
 - POS – Possessive ending (' s)
 - CD – Cardinal number (one , 2)
 - LS – List item marker (e.g., 1. in a list)
-

```
In [2]: #Let start with begins
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
import re
from nltk import pos_tag

corpus= """
Google is a multinational technology company that specializes in Internet-related services and products. It was founded in 1998.

In 2006, Google acquired YouTube, now the world's largest video-sharing platform. The company also developed the Android operating system.

In 2015, Google underwent a major corporate restructuring and became a subsidiary of Alphabet Inc., a new parent company created to manage Google's various businesses.

With a mission to "organize the world's information and make it universally accessible and useful," Google has deeply influenced the world.

Let me know if you want this rewritten in Tamil or expanded into multiple paragraphs for a report or presentation!

"""

sent=sent_tokenize(corpus)
type(sent)

words=[]
for i in range(0,len(sent)):
    for word in sent:
        review=re.sub('[^a-zA-Z]', " ",word)
        word=word_tokenize(review)
        words.append(word)

words
```

```
Out[2]: [['Google',
'is',
'a',
'multinational',
'technology',
'company',
'that',
'specializes',
'in',
'Internet',
'related',
'services',
'and',
'products'],
['It',
'was',
'founded',
'in',
'by',
..],
..]
```

```
In [3]: tag=[]
for char in words:
    tags=pos_tag(char)
    tag.append(tags)
```

Named Entity Recognition (NER):

Named Entity Recognition (NER) is a subtask of Natural Language Processing (NLP) that focuses on identifying and classifying named entities in a text into predefined categories.

In [4]: tag

```
Out[4]: [(('Google', 'NNP'),
          ('is', 'VBZ'),
          ('a', 'DT'),
          ('multinational', 'JJ'),
          ('technology', 'NN'),
          ('company', 'NN'),
          ('that', 'WDT'),
          ('specializes', 'VBZ'),
          ('in', 'IN'),
          ('Internet', 'NNP'),
          ('related', 'JJ'),
          ('services', 'NNS'),
          ('and', 'CC'),
          ('products', 'NNS')),
          (('It', 'PRP'),
          ('was', 'VBD'),
          ('founded', 'VBN'),
          ('in', 'IN'),
          ('by', 'IN'),
          ('', ''))]
```

```
In [5]: from nltk import ne_chunk
list_N=[]
for i in tag:
    NER=ne_chunk(i)
    list_N.append(NER)
```

In [6]: list_N[5]

```
Out[6]: Tree('S', [(('The', 'DT'), ('company', 'NN'), ('also', 'RB'), ('developed', 'VBD'), ('the', 'DT'), Tree('ORGANIZATION', [(('Android', 'NNP')], ('operating', 'NN'), ('system', 'NN'), ('which', 'WDT'), ('powers', 'VBZ'), ('the', 'DT'), ('majority', 'NN'), ('of', 'IN'), ('smartphones', 'NNS'), ('globally', 'RB'))])])])
```

```
In [64]: pip install svgling
#use for display the tree
```

Requirement already satisfied: svgling in c:\users\vicky\anaconda3\lib\site-packages (0.5.0)
 Requirement already satisfied: svgwrite in c:\users\vicky\anaconda3\lib\site-packages (from svgling) (1.4.3)
 Note: you may need to restart the kernel to use updated packages.

In [7]: list_N[50]

```
Out[7]: Tree('S', [(('The', 'DT'), ('company', 'NN'), ('s', 'VBD'), ('core', 'JJR'), ('product', 'NN'), ('the', 'DT'), Tree('ORGANIZATION', [(('Google', 'NNP'), ('Search', 'NNP'))], ('engine', 'NN'), ('quickly', 'RB'), ('became', 'VBD'), ('the', 'DT'), ('most', 'RBS'), ('widely', 'RB'), ('used', 'VBN'), ('search', 'NN'), ('tool', 'NN'), ('in', 'IN'), ('the', 'DT'), ('world', 'NN'), ('due', 'JJ'), ('to', 'TO'), ('its', 'PRP$'), ('speed', 'NN'), ('relevance', 'NN'), ('and', 'CC'), ('simplicity', 'NN'))])])
```

Bag of words

What is Bag of Words?

Bag of Words (BoW) is a simple and commonly used technique to represent text data in a way that can be used by machine learning models.

It ignores grammar and word order.

Focuses only on (word occurrence) — how many times each word appears in the text.

1.Build the Vocabulary

sentence

1.Hi i am vigneshwaran 2.data science is booming tech 3.welcome to AI world

After stopword

using stopword and collect the vocabulary

[hi,vigneshwara,data,science,booming,tech,welcome ,AI,words]

important note

Bows techniuqe using Frequence based word in vocabulary

Advantage of Bow

- ✓ simple and Intuitive
- ✓ fixed sized i/o -> ML Algorithm

disadvantage of Bow

- ✗ sparse matrix or array (more 0 and 1)->overfitting
- ✗ ordering of the word is getting changed
- ✗ out of Vocabulary(in test)
- ✗ sementic meaning not capure (but than one-hot_encoder)

② Bag of Words

Dataset

Text	O/p
He is a good boy	1
She is a good girl	1
Boy and girl are good	1

Lower all the words case
Stopwords

S1 → good boy
S2 → good girl
S3 → Boy girl good

good
boy
girl

Vocabulary frequency

Vocabulary	frequency	[good	boy	girl]	O/p		
good	3	⇒ S1	[1	1	0]	1	
boy	2		S2	[1	0	1]	1
girl	2		S3	[1	1	1]	1

Binary Bow and Bow

{ 1 and 0 } { Count will get updated based on frequency }

Bag-of-Words Model Bag-of-words model is a way of representing text data when modeling text with machine learning algorithms. Machine learning algorithms cannot work with raw text directly; the text must be converted into well defined fixed-length(vector) numbers.

```
In [12]: from sklearn.feature_extraction.text import CountVectorizer

#multiple document

text=["It was the best of times","it was the worst of times","it was the age of wisdom","it was the age of foolishness"]

#create the transform
vectorize=CountVectorizer()

vectorize.fit(text)
```

```
In [16]: print(sorted(vectorize.vocabulary_))

['age', 'best', 'foolishness', 'it', 'of', 'the', 'times', 'was', 'wisdom', 'worst']
```

```
In [32]: vector=vectorize.transform(text)
print(vectorize.vocabulary_)

{'it': 3, 'was': 7, 'the': 5, 'best': 1, 'of': 4, 'times': 6, 'worst': 9, 'age': 0, 'wisdom': 8, 'foolishness': 2}
```

```
In [23]: vector.shape
```

```
Out[23]: (4, 10)
```

```
In [24]: vector.toarray()
```

```
Out[24]: array([[0, 1, 0, 1, 1, 1, 1, 1, 0, 0],
               [0, 0, 0, 1, 1, 1, 1, 1, 0, 1],
               [1, 0, 0, 1, 1, 1, 0, 1, 1, 0],
               [1, 0, 1, 1, 1, 1, 0, 1, 0, 0]], dtype=int64)
```

```
In [31]: text2=["the the the times"]
text3=["what is your name"]
vector=vectorize.transform(text3)
print(vector.toarray())
vector.shape
```

```
[[0 0 0 0 0 0 0 0 0 0]]
```

```
Out[31]: (1, 10)
```

Using Dataset

```
In [3]: import pandas as pd

df=pd.read_csv("SMSSpamCollection.txt", sep="\t", names=["ham/spam", "messages"])
```

```
In [4]: df
```

```
Out[4]:
```

	ham/spam	messages
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like I'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
In [5]: #Goal is messages--->vectore

import re #regulare expression
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
stemming=SnowballStemmer("english")
from nltk.corpus import stopwords
```

```
In [6]: corpus=[]
for i in range(0,len(df)):
    review=re.sub('[^a-zA-Z]'," ",df["messages"][i])
    review=review.lower()
    review=review.split()
    review=[stemming.stem(words) for words in review if not words in set(stopwords.words("english"))]
    review=" ".join(review)
    corpus.append(review)
```

corpus

```
[ 'go jurong point crazi avail bugi n great world la e buffet cine got amor wat',  
  'ok lar joke wif u oni',  
  'free entri wkli comp win fa cup final tkts st may text fa receiv entri question std txt rate c appli',  
  'u dun say earli hor u c already say',  
  'nah think goe usf live around though',  
  'freemsg hey darl week word back like fun still tb ok xxx std chgs send rcv',  
  'even brother like speak treat like aid patent',  
  'per request mell mell oru minnaminungint nurungu vettam set callertun caller press copi friend callertun',  
  'winner valu network custom select receivea prize reward claim call claim code kl valid hour',  
  'mobil month u r entitl updat latest colour mobil camera free call mobil updat co free',  
  'gonna home soon want talk stuff anymor tonight k cri enough today',  
  'six chanc win cash pound txt csh send cost p day day tsandc appli repli hl info',  
  'urgent week free membership prize jackpot txt word claim c www dbuk net lcltld pobox ldnw rw',  
  'search right word thank breather promis wont take help grant fulfil promis wonder bless time',  
  'date sunday',  
  'xxxmobilemovieclub use credit click wap link next txt messag click http wap xxxmobilemovieclub com n qjkgighjjgcbl',  
  'oh k watch',  
  'eh u rememb spell name yes v naughti make v wet',  
  'fine way u feel way gota b',
```

```
#bag of word
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
#initize
vectorizer=CountVectorizer(max_features=2000,binary=True,ngram_range=(1,3))
#max_fearure 2000 top frequency words
#Binary=True only 1 or 1
#to fit and train the model
x=vectorizer.fit_transform(corpus).toarray()
```

```
np.set_printoptions(edgeitems=30, linewidth=100000,  
    formatter=dict(float=lambda x: "%.3g" % x))
```

[illegible]

```
len(vectorizer.vocabulary_)
```

2000

```
vectorizer.vocabulary_
```

```
{ 'go': 644,  
  'point': 1295,  
  'crazi': 362,  
  'avail': 94,  
  'bugi': 180,  
  'great': 691,  
  'world': 1944,  
  'la': 873,  
  'cine': 284,  
  'got': 682,  
  'wat': 1863,  
  'ok': 1186,  
  'lar': 883,  
  'joke': 846,  
  'wif': 1913,  
  'free': 584,  
  'entri': 503,  
  'wkli': 1935,  
  'comp': 321,  
  'i': 1010
```

N_gram

Formal Definition:

An N-gram is a subsequence of N tokens extracted from a longer sequence in natural language processing (NLP), where N denotes the number of elements in the group.

ngram_range	Captures	When to Use
(1,1)	Individual words (unigrams)	Basic text analysis, bag-of-words models
(1,2)	Unigrams + Bigrams	More context, short phrases
(2,2)	Only Bigrams	Sequence-based tasks
(1,3)	Unigrams + Bigrams + Trigrams	Rich context, but high dimensionality
(3,3)	Only Trigrams	Grammar-like patterns (good for stylometry)

① N-grams Eg: bigrams, trigrams

$S_1 \rightarrow$ The food is good
 $S_2 \rightarrow$ The food is not good

Bigram

	food	not	good	food good	food not	not good
S_1	1	0	1	1	0	0
S_2	1	1	1	0	1	1

$S_{klearn} \rightarrow n\text{-grams} = (1,1) \rightarrow \text{unigrams}$
 $= (1,2) \rightarrow \text{unigram, bigram}$
 $= (1,3) \rightarrow \text{unigram, bigram, trigram}$
 $= (2,3) \rightarrow \text{Bigram, trigram}$

Unigram ngram_range=(1,1)

In [15]: vectorizer.vocabulary_

Out[15]:

```
{'go': 674,
'point': 1287,
'crazi': 388,
'avail': 125,
'bugi': 237,
'great': 698,
'world': 1946,
'la': 927,
'cine': 314,
'got': 689,
'wat': 1883,
'ok': 1197,
'lar': 935,
'joke': 893,
'wif': 1916,
'oni': 1205,
'free': 612,
'entri': 515,
'wkli': 1936,
'...': 240}
```

Bigrams ngram_range=(1,2)

In [21]: vectorizer.vocabulary_

```
Out[21]: {'go': 644,
'point': 1295,
'crazi': 362,
'avail': 94,
'bugi': 180,
'great': 691,
'world': 1944,
'la': 873,
'cine': 284,
'got': 682,
'wat': 1863,
'ok': 1186,
'lar': 883,
'joke': 846,
'wif': 1913,
'free': 584,
'entri': 503,
'wkli': 1935,
'comp': 321,
'...': 1000}
```

trigram ngram_range=(1,3)

In [26]: vectorizer.vocabulary_

```
Out[26]: {'go': 652,
'point': 1300,
'crazi': 362,
'avail': 86,
'bugi': 173,
'great': 684,
'world': 1946,
'la': 873,
'cine': 270,
'got': 679,
'wat': 1872,
'ok': 1191,
'lar': 885,
'joke': 846,
'wif': 1914,
'free': 590,
'entri': 503,
'wkli': 1937,
'comp': 315,
'...': 1000}
```

◆ TF-IDF: Term Frequency – Inverse Document Frequency

TF-IDF is a statistical measure used in Natural Language Processing to evaluate how important a word is to a document in a collection (corpus).

✓ 1. Definition TF (Term Frequency): How often a word appears in a document.

IDF (Inverse Document Frequency): How unique or rare the word is across all documents.

What is TF-IDF? TF-IDF is an improved version of BoW. It tries to reduce the impact of common words (like "the", "is", "vicky") and highlight important words in a document.

TF = Term Frequency How often a word appears in a document

IDF = Inverse Document Frequency Penalizes words that appear in many documents (less informative)

TF-IDF (t, d) = $TF(t, d) \times \log (\frac{N}{DF(t)})$ TF-IDF(t,d)=TF(t,d)*log(DF(t) N) Where:

t : term (word)

d : document

N : total number of documents

$DF(t)$: number of documents with term t

◆ Example: "AI" might appear in every document, so its IDF is low "Streamlit" appears in only a few, so it gets higher weight

④ TF-IDF [Term Frequency - Inverse Document Frequency]

$S_1 \rightarrow \text{good boy}$
 $S_2 \rightarrow \text{good girl}$
 $S_3 \rightarrow \text{boy girl good}$

$\text{Term Freq(TF)} = \frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$
 $\text{IDF} = \log_e \left(\frac{\text{No. of sentences}}{\text{No. of sentences containing the word}} \right)$

	Term Frequency			*	IDF	
	S_1	S_2	S_3		Words	IDF
good	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$		good	$\log_e(3/\frac{1}{3}) = 0$
boy	$\frac{1}{2}$	0	$\frac{1}{3}$		boy	$\log_e(3/\frac{1}{2})$
girl	0	$\frac{1}{2}$	$\frac{1}{3}$		girl	$\log_e(3/\frac{1}{2})$

Final TF-IDF

	[good]	[boy]	[girl]	$\frac{0}{1}$	Bow
Sent 1	0	$\frac{1}{2} \times \log_e(3/\frac{1}{2})$	0		1 1 0
Sent 2	0	0	$\frac{1}{2} \log_e(3/\frac{1}{2})$		1 0 1
Sent 3	0	$\frac{1}{3} \log_e(3/\frac{1}{2})$	$\frac{1}{3} \log_e(3/\frac{1}{2})$		1 1 1

Advantages

1. Intuitive
2. fixed size $\rightarrow$ vocab size
3. word importance is getting captured

Disadvantages

1. sparsity still exists
2. out of vocabulary

```
In [37]: import pandas as pd
df1=pd.read_csv("amazon_reviews.csv")
```

In [50]: df1

Out[50]:

	Unnamed: 0	reviewerName	overall	reviewText	reviewTime	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating	w
0	0	NaN	4.0	No issues.	2014-07-23	138	0	0	0	0	0.0	
1	1	Omie	5.0	Purchased this for my device, it worked as adv...	2013-10-25	409	0	0	0	0	0.0	
2	2	1K3	4.0	it works as expected. I should have sprung for...	2012-12-23	715	0	0	0	0	0.0	
3	3	1m2	5.0	This think has worked out great.Had a diff. br...	2013-11-21	382	0	0	0	0	0.0	
4	4	2&1/2Men	5.0	Bought it with Retail Packaging, arrived legit...	2013-07-13	513	0	0	0	0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...
4910	4910	ZM "J"	1.0	I bought this Sandisk 16GB Class 10 to use wit...	2013-07-23	503	0	0	0	0	0.0	
4911	4911	Zo	5.0	Used this for extending the capabilities of my...	2013-08-22	473	0	0	0	0	0.0	
4912	4912	Z S Liske	5.0	Great card that is very fast and reliable. It ...	2014-03-31	252	0	0	0	0	0.0	
4913	4913	Z Taylor	5.0	Good amount of space for the stuff I want to d...	2013-09-16	448	0	0	0	0	0.0	
4914	4914	Zza	5.0	I've heard bad things about this 64gb Micro SD...	2014-02-01	310	0	0	0	0	0.0	

4915 rows × 13 columns

In [81]: df1.dtypes

Out[81]:

```

Unnamed: 0          int64
reviewerName       object
overall            float64
reviewText         object
reviewTime         object
day_diff           int64
helpful_yes        int64
helpful_no         int64
total_vote         int64
score_pos_neg_diff int64
score_average_rating float64
wilson_lower_bound float64
text               string[python]
dtype: object

```

In [94]: df1["text"].dtypes

Out[94]: string[python]

```

In [80]: # Convert the 'text' column to pandas' string dtype
df1['text'] = df1['text'].astype('string')

# Check the new data type
print(df1['text'].dtype) # Should show dtype('string')

```

string

In [91]:

df1.shape

Out[91]: (4915, 13)

In [89]:

```
import re
from nltk.stem import PorterStemmer
stemmer1=PorterStemmer()

cleanTheText=[]

for i in range(0,len(df1)):
    review=re.sub('[^a-zA-Z]', " ", df1["text"][i])
    review=review.lower()
    review=review.split()
    review=[stemmer1.stem(words) for words in review if not words in set(stopwords.words("english"))]
    review=" ".join(review)
    cleanTheText.append(review)
```

In [93]:

len(cleanTheText)

Out[93]: 4915

In [103]:

```
#using
# from sklearn.feature_extraction.text import tfidfvectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

TFIDF=TfidfVectorizer(max_features=3000,ngram_range=(1,2))

vectore=TFIDF.fit_transform(cleanTheText)
```

In [108]:

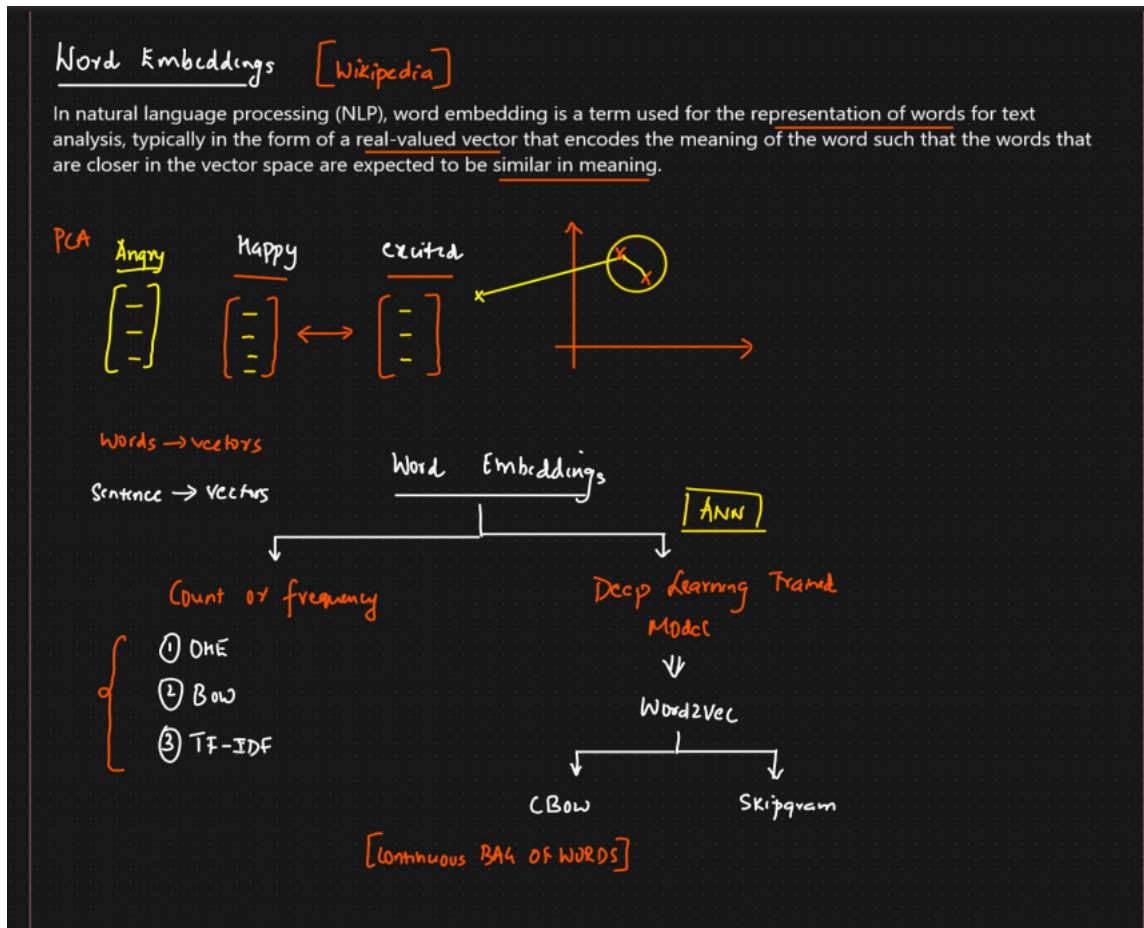
TFIDF.vocabulary_

Out[108]:

```
{'issu': 1308,
 'purchas': 2045,
 'devic': 651,
 'work': 2902,
 'advertis': 47,
 'never': 1664,
 'much': 1605,
 'phone': 1838,
 'memori': 1516,
 'sinc': 2382,
 'download': 689,
 'lot': 1438,
 'stuff': 2541,
 'brainer': 228,
 'devic work': 657,
 'work advertis': 2903,
 'phone memori': 1863,
 'lot stuff': 1448,
 'expect': 800,
 'text': 1330}
```

Word Embeddings

In natural lanugae processing(NLP). word embedding is a term used for the analysis ,typicall in the form of a real-valued vector the encodes the meaning of the word such that the words tthat are closer in the vectore space aare expected to be similiary in meaning.



word embedding

✓ count or frequency

1.OHE (one hot encoder) 2.Bow (bag of words) 4.TF_IDF (Term frequency and inverse document frequency)

✓ Deep learning Trend model 1.Word2vec !) cbow !!) Skipgram

Word2vec (♦ Important concept ♦)

Word Embedding Techniques

In natural language processing, **word embedding** means converting words or text into numeric vector representations that computers can process. Early methods used simple counts or binary features, while modern methods use neural networks to learn dense vectors. Count-based methods (like one-hot, bag-of-words, TF-IDF) produce very high-dimensional, sparse vectors, whereas neural methods (like Word2Vec) produce lower-dimensional embeddings that capture word meanings. Each approach has trade-offs in complexity, meaning-capture, and typical use cases.

Traditional (Count-Based) Methods

One-Hot Encoding (OHE)

One-hot encoding represents each word as a binary vector with one "1" and the rest "0." For example, with vocabulary ["cat", "dog", "bird"], the word "dog" would be encoded as $[0, 1, 0]$: a 1 at the position for "dog" and 0 elsewhere. This is simple and easy to understand, but the vectors are **very large** (one dimension per word) and **sparse**. In practice this means millions of dimensions if the vocabulary is large. One-hot vectors also carry no information about word meaning or similarity (all words are "equally far" in this space).

- **Advantages:** Very simple; creates unique feature per word.
- **Disadvantages:** Vector dimension = vocabulary size (often hundreds of thousands); extremely sparse; no notion of word similarity.

Because of these limitations, one-hot encoding is seldom used alone for modeling text beyond toy examples.

Bag-of-Words (BoW)

The **bag-of-words** model represents a document by the counts of each word in it, ignoring order. For example, two short documents:

- Doc1: "apple banana banana"
- Doc2: "banana orange apple"

Using vocabulary [apple, banana, orange], BoW vectors are:

- Doc1 $\rightarrow [1, 2, 0]$ (1 apple, 2 bananas, 0 oranges)
- Doc2 $\rightarrow [1, 1, 1]$ (1 apple, 1 banana, 1 orange)

BoW is called a “bag” because it **discards word order**, focusing only on which words appear and how often. This yields a fixed-length vector for each document that can be used in classification or clustering. BoW features have been **widely used in text classification**, spam detection, and information retrieval.

- **Advantages:** Simple and intuitive; turns text into fixed-size numeric features (word counts). It often works well in classical models (e.g. Naive Bayes, SVM).
- **Disadvantages:** Ignores word order and context (so “dog bites man” and “man bites dog” look similar). The vectors are typically **very sparse and high-dimensional**, making computation and modeling harder. Very common words (like “the” or “and”) dominate counts unless handled. The vocabulary size must be managed carefully to control sparsity.

TF-IDF (Term Frequency–Inverse Document Frequency)

TF-IDF builds on BoW by scaling each word’s count by how unique the word is across all documents. **Term Frequency (TF)** measures how often a word appears in a document (often normalized by document length). **Inverse Document Frequency (IDF)** down-weights words that appear in many documents. In effect, common words like “the” get low weight, while rare but important words get higher weight. The TF-IDF value for a word in a document is typically the product of its TF and IDF scores.

For example, if “apple” appears 3 times in a short document but only in 1 out of 1000 documents total, its TF is high and IDF is large, giving a high TF-IDF score for that document. Conversely, a common word like “is” appearing in almost every document would get a low IDF, lowering its overall TF-IDF weight.

- **Advantages:** Still fairly simple, but highlights important words while reducing the impact of common “stop words.” TF-IDF vectors often improve performance in search and classification compared to raw counts.
- **Disadvantages:** Like BoW, TF-IDF ignores word order and semantics. Vectors remain high-dimensional and sparse (one entry per vocabulary word). Computing IDF requires processing the whole corpus.

Typical use cases: TF-IDF is a cornerstone of information retrieval and text mining; many search engines and recommender systems use TF-IDF features for ranking or as input to models. It is a common baseline for document classification or clustering.

Word2Vec (Neural Embeddings)

While count-based methods rely on raw frequencies, **Word2Vec** is a neural-network approach that learns dense, low-dimensional vectors for words from large text corpora. Introduced by Mikolov et al. (2013) at Google, Word2Vec trains a shallow (two-layer) network to predict word contexts. After training, each word is assigned a fixed-size vector (e.g. 100–300 dimensions) such that words with similar contexts end up with similar vectors. For example, “king” and “queen” will have vectors close to each other, and arithmetic relations like $king - man + woman \approx queen$ tend to emerge. In these models, context (nearby words) is the key signal, embodying the idea that words appearing in similar contexts have related meanings.

Word2Vec comes in two main variants:

Continuous Bag-of-Words (CBOW)

The CBOW model predicts a **target word** from its surrounding **context words**. Imagine a sentence: “the **cat** sat on the mat.” With a window size of 2, the context words around “sat” are {“the”, “cat”, “on”, “the”}. CBOW takes (for example) the words “the”, “cat”, “on”, “the” as input and tries to predict the target “sat.” In practice, the context word vectors are averaged (or summed) to predict the target. The order of context words is ignored (a bag-of-words assumption for context).

Example: Suppose the sentence is “I like to eat oranges”, and we focus on target “eat” with window size 1. The context is {“to”}. CBOW trains a model that takes “to” as input and tries to predict “eat”.

- **Advantages:** CBOW is fast to train and works well for frequent words. It tends to be simpler since it predicts the center word from an average of context.
- **Disadvantages:** It can be less effective for rare words because averaging context can dilute signals.

According to the original Word2Vec authors, CBOW is generally *faster* to train while sacrificing some performance on infrequent words. CBOW is often used when training speed is a priority and when you have a very large corpus.

Skip-gram

The skip-gram model flips CBOW around: it uses the **target word to predict its context words**. In our sentence “the cat sat on the mat,” take “sat” as the target (input) and try to predict its neighbors {“cat”, “on”} (using a window of size 2, ignoring the repeated “the”). Concretely, skip-gram forms training pairs like (“sat” → “cat”) and (“sat” → “on”) and tries to maximize the probability of the correct context words given the target. The network has one input word and tries to predict one context word at a time, looping over each context word.

Example: Using the window example above, skip-gram would train on pairs: (target=“eat”, context=“to”) and (target=“eat”, context=“oranges”). Each pair is a training example for the model to learn that “eat” is often found near “to” and “oranges.”

- **Advantages:** Skip-gram often gives better representations for **rare words** because each word predicts many context words (more training signal per word). It explicitly models each neighbor, which can capture asymmetry (order has a slight effect via weighing).
- **Disadvantages:** It is slower to train than CBOW because it processes each context word separately.

The skip-gram architecture tends to **weight nearer context words more heavily** and was found to perform better on infrequent words. It is commonly used when one wants high-quality embeddings and can afford more training time.

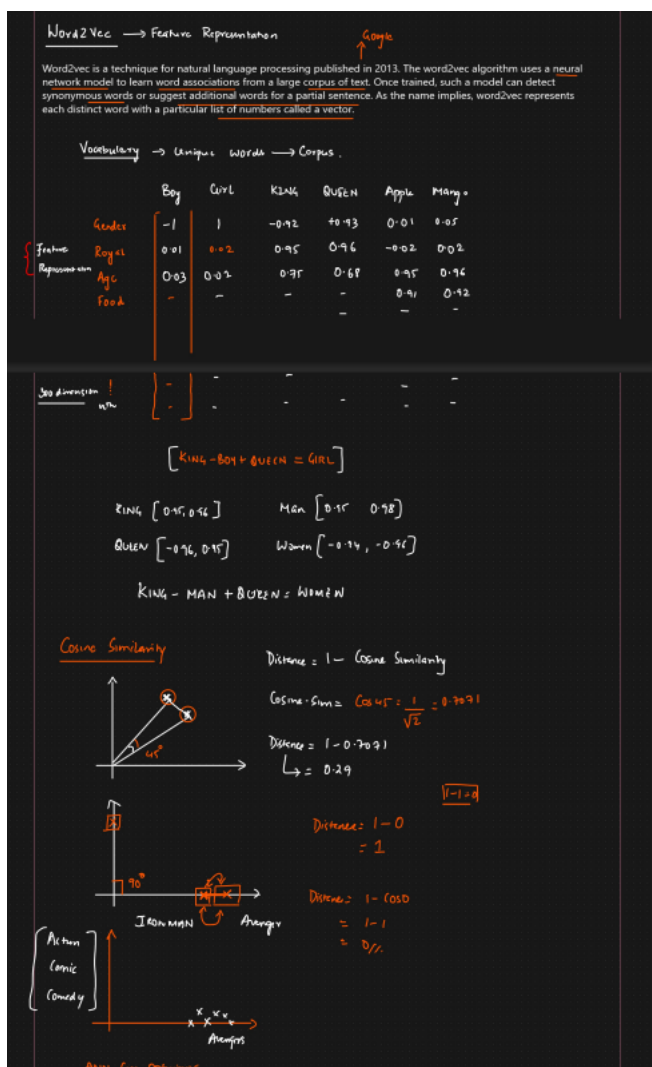
After training either model, the learned word vectors (the hidden-layer weights) encode semantic relationships: words that share contexts end up close in the embedding space. For instance, *Mexico* + *D.C.* often lies near *United States* in embedding space. These dense vectors capture both semantic and syntactic information from the text corpus.

Summary and Use-Cases

- **One-Hot Encoding** is a simple, sparse representation (vector per word). It is only practical for very small vocabularies or as a starting point; it does **not** capture any notion of word meaning.

- In practice, simple count methods (BoW/TF-IDF) are still used for straightforward tasks and baselines, especially when interpretability or ease of use is important. Word2Vec-style embeddings are used when semantics matter and enough text data is available, often as a foundation for modern NLP systems. Each method has its niche depending on data size, computation budget, and the need to capture meaning.

Sources: Authoritative tutorials and references on word representations and Word2Vec.



Gensim

```
In [2]: import pandas as pd
import numpy as np
import re
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from gensim.models import KeyedVectors
import gensim.downloader as api
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
In [13]: # Load NLTK resources
# nltk.download('punkt')
# nltk.download('stopwords')
```

```
In [12]: # from gensim.models import KeyedVectors

# print("Loading Word2Vec model...")
# w2v = KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin", binary=True)
# print("Loaded successfully!")

# print(w2v['king']) # See the vector
```

```
In [3]: import gensim.downloader as api
w2v=api.load('word2vec-google-news-300')
```

```
In [4]: len(w2v["king"])
```

```
Out[4]: 300
```

```
In [5]: url = "https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv"
df=pd.read_csv(url,sep="\t",names=["spam/ham", "messages"])
```

```
In [6]: df
```

```
Out[6]:
```

	spam/ham	messages
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
In [7]: def preprocess(text):
review=re.sub(f"[{re.escape(string.punctuation)}]", "", text)
review=re.sub('[^a-zA-Z]', " ", review)
review=review.lower()
review=word_tokenize(review)
return [w for w in review if w not in set(stopwords.words("english"))]
```

```
In [8]: df['tokens']=df['messages'].apply(preprocess)
```

```
In [9]: df
```

```
Out[9]:
```

	spam/ham	messages	tokens
0	ham	Go until jurong point, crazy.. Available only ...	[go, jurong, point, crazy, available, bugis, n...
1	ham	Ok lar... Joking wif u oni...	[ok, lar, joking, wif, u, oni]
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, wkly, comp, win, fa, cup, final,...
3	ham	U dun say so early hor... U c already then say...	[u, dun, say, early, hor, u, c, already, say]
4	ham	Nah I don't think he goes to usf, he lives aro...	[nah, dont, think, goes, usf, lives, around, t...
...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	[nd, time, tried, contact, u, u, pound, prize,...
5568	ham	Will ü b going to esplanade fr home?	[b, going, esplanade, fr, home]
5569	ham	Pity, * was in mood for that. So...any other s...	[pity, mood, soany, suggestions]
5570	ham	The guy did some bitching but I acted like i'd...	[guy, bitching, acted, like, id, interested, b...
5571	ham	Rofl. Its true to its name	[rofl, true, name]

5572 rows × 3 columns

```
In [10]: df["spam/ham"] = df["spam/ham"].map({'spam':1, 'ham':0})
```

```
In [11]: df["spam/ham"].value_counts()
```

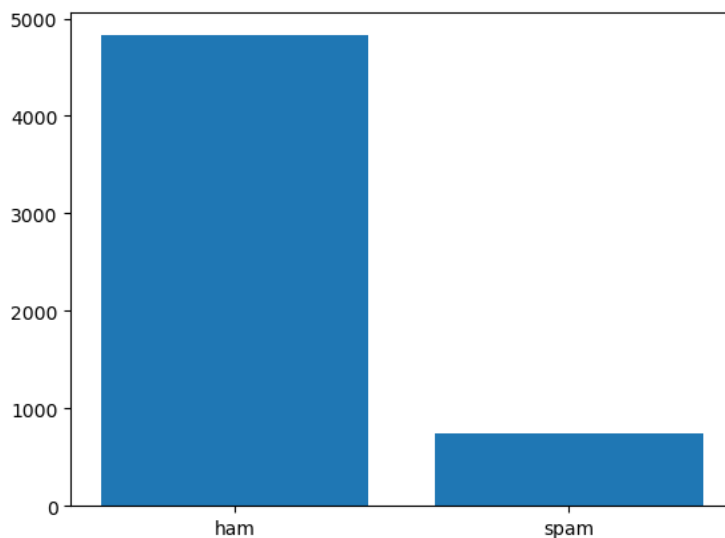
```
Out[11]: spam/ham
0      4825
1       747
Name: count, dtype: int64
```

```
In [12]: df["spam/ham"].unique
values=df["spam/ham"].value_counts()
```

```
In [13]: import matplotlib.pyplot as plt
labels=["ham", "spam"]
counts=values.values.tolist()
```

```
In [14]: fig, ax=plt.subplots()
ax.bar(labels, counts)
```

```
Out[14]: <BarContainer object of 2 artists>
```



word2vec and Average word2vec vectore

```
In [15]: # For gensim KeyedVectors (e.g., GoogleNews-vectors-negative300.bin)
print(list(w2v.key_to_index.keys())[:50]) # Show first 50 words
```

```
['</s>', 'in', 'for', 'that', 'is', 'on', '##', 'The', 'with', 'said', 'was', 'the', 'at', 'not', 'as', 'it', 'be', 'from',
'by', 'are', 'I', 'have', 'he', 'will', 'has', '####', 'his', 'an', 'this', 'or', 'their', 'who', 'they', 'but', '$', 'ha
d', 'year', 'were', 'we', 'more', '###', 'up', 'been', 'you', 'its', 'one', 'about', 'would', 'which', 'out']
```

```
In [39]: def get_vector(tokens):
    vector_size = w2v.vector_size
    vec=[ w2v[word] for word in tokens if word in w2v]
    return np.mean(vec,axis=0) if vec else np.zeros(vector_size)
```

```
In [40]: vectore_size = w2v.vector_size
print(vectore_size)
```

```
300
```

```
In [41]: # Now apply it to your 'tokens' column
X = df["tokens"].apply(lambda tokens: get_vectore(tokens))
X
```

```
Out[41]: 0      [-0.019805908, 0.05167062, 0.02709961, 0.21868...
1      [-0.06323496, 0.0803833, 0.060943604, 0.102498...
2      [-0.009838778, -0.016867245, -0.06924977, 0.07...
3      [-0.06568061, 0.0262146, 0.1081543, 0.0869751,...
4      [0.041534424, 0.02865982, 0.056930542, 0.16627...
...
5567   [-0.012015207, 0.04544435, 0.02331434, 0.06880...
5568   [0.021606445, 0.090625, 0.109191895, 0.0634216...
5569   [0.044270832, 0.15771484, 0.0152994795, 0.0895...
5570   [0.11360387, 0.016531808, -0.014187677, 0.0879...
5571   [0.111979164, 0.0679423, 0.087239586, 0.075764...
Name: tokens, Length: 5572, dtype: object
```

```
In [58]: X = np.vstack(df['vectore'].values)
```

```
In [ ]:
```

```
In [ ]:
```

```
In [42]: df["vectore"]=df["tokens"].apply(get_vectore)
```

```
In [43]: df
```

```
Out[43]:
```

	spam/ham	messages	tokens	vectore
0	0	Go until jurong point, crazy.. Available only ...	[go, jurong, point, crazy, available, bugis, n...	[-0.019805908, 0.05167062, 0.02709961, 0.21868...
1	0	Ok lar... Joking wif u oni...	[ok, lar, joking, wif, u, oni]	[-0.06323496, 0.0803833, 0.060943604, 0.102498...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, wkly, comp, win, fa, cup, final,...	[-0.009838778, -0.016867245, -0.06924977, 0.07...
3	0	U dun say so early hor... U c already then say...	[u, dun, say, early, hor, u, c, already, say]	[-0.06568061, 0.0262146, 0.1081543, 0.0869751,...
4	0	Nah I don't think he goes to usf, he lives aro...	[nah, dont, think, goes, usf, lives, around, t...	[0.041534424, 0.02865982, 0.056930542, 0.16627...
...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	[nd, time, tried, contact, u, u, pound, prize,...	[-0.012015207, 0.04544435, 0.02331434, 0.06880...
5568	0	Will ù b going to esplanade fr home?	[b, going, esplanade, fr, home]	[0.021606445, 0.090625, 0.109191895, 0.0634216...
5569	0	Pity, * was in mood for that. So...any other s...	[pity, mood, soany, suggestions]	[0.044270832, 0.15771484, 0.0152994795, 0.0895...
5570	0	The guy did some bitching but I acted like i'd...	[guy, bitching, acted, like, id, interested, b...	[0.11360387, 0.016531808, -0.014187677, 0.0879...
5571	0	Rofl. Its true to its name	[rofl, true, name]	[0.111979164, 0.0679423, 0.087239586, 0.075764...

5572 rows × 4 columns

```
In [126]: # x=np.vstack(df['vectore'].values)
          df['vectore']][1]
```

```
Out[126]: array([-0.06323496,  0.0803833 ,  0.0609436 ,  0.10249837, -0.07885742,
  0.03523763, -0.05303955, -0.07343546,  0.01005046,  0.08062744,
 -0.0195516 , -0.1706543 , -0.2672526 ,  0.05753581, -0.16577148,
  0.14929199,  0.04094442,  0.06374105, -0.0362498 , -0.07157389,
  0.00939941,  0.01387533,  0.30395508, -0.05476888, -0.12422689,
 -0.03968303, -0.15987141, -0.07067871,  0.0390625 , -0.10019938,
 -0.06841024,  0.07425944,  0.04516602, -0.146403 , -0.13387044,
 -0.05900065,  0.04672241,  0.11028036,  0.09409078,  0.06640625,
  0.06648763, -0.20076497,  0.24143474,  0.03127034,  0.08467611,
 -0.07048544,  0.01353327, -0.17317708, -0.10843913,  0.07314046,
 -0.15913899,  0.1184082 ,  0.14880371,  0.14562988,  0.05458577,
  0.10131836, -0.12984212, -0.14930598,  0.07944743, -0.24210612,
 -0.07764053,  0.10290527, -0.08703613, -0.02243551, -0.00543213,
 -0.18188477, -0.09377035, -0.12186686, -0.14912923,  0.08772787,
  0.02494558,  0.09077962, -0.00662947,  0.09547678, -0.23234431,
  0.05001163,  0.09655762,  0.04081662,  0.00406901, -0.07511393,
 -0.12789917, -0.02884928, -0.03690593, -0.10479736, -0.00889079,
  0.1167806 , -0.04044596,  0.2351888 , -0.02487055, -0.01760864,
  0.00500488,  0.03232574, -0.06341553, -0.07454427, -0.08085123,
  0.07983398,  0.02467855, -0.00491333,  0.19685872, -0.10754395,
 -0.1861674 ,  0.08357748,  0.03401693, -0.01993815, -0.00610352,
  0.11409505, -0.03096517, -0.03019206,  0.09175619,  0.00622559,
 -0.12837727, -0.20113118,  0.03507487, -0.07635244, -0.00262451,
 -0.02945963,  0.05289714, -0.18933105, -0.09956869, -0.09684245,
  0.0053304 ,  0.07983398, -0.04915364, -0.03940837,  0.09041341,
 -0.0594991 , -0.18025716,  0.01475525,  0.08677164, -0.01289876,
 -0.23673503, -0.04939779, -0.01319377,  0.05969238, -0.1644694 ,
  0.090271 , -0.07374064,  0.0674642 ,  0.18359375,  0.0447998 ,
  0.2265625 , -0.08836874,  0.05385335, -0.05008952, -0.02142334,
  0.02649943, -0.13080852, -0.0609436 ,  0.03885905, -0.14682007,
  0.1282959 , -0.07084147, -0.13535564, -0.10961914, -0.08219401,
 -0.07283529, -0.03085327,  0.17177708,  0.04867554, -0.0850741 ,
 -0.0061849 ,  0.07263184,  0.1324056 , -0.03033447,  0.02278646,
 -0.206604 ,  0.13596599,  0.05786133, -0.13761394,  0.04938762,
 -0.03634644,  0.01741537,  0.04140218,  0.06703695, -0.08384196,
  0.11649577,  0.15710449, -0.16137695,  0.16007487, -0.04133097,
  0.05829875,  0.04601542,  0.10113525,  0.07564291,  0.0338033 ,
  0.14013672, -0.14217122, -0.01390076, -0.04642741, -0.04865519,
  0.11712646,  0.03525798, -0.03574626,  0.00050863,  0.14066188,
  0.05088806, -0.09578451, -0.00535075, -0.04427083, -0.02101644,
 -0.13963191,  0.01772054, -0.15555827, -0.08818563,  0.05900065,
 -0.11566671, -0.04836019,  0.02021408, -0.1270345 ,  0.09380087,
  0.11279297,  0.03533936, -0.13126628, -0.04351457, -0.03907267,
  0.04368083,  0.02048747, -0.08548991, -0.18815105,  0.00562541,
 -0.0954183 ,  0.06510416,  0.11675008,  0.03075663,  0.08715311,
 -0.12009684,  0.02762858,  0.03179932, -0.13682048, -0.1673177 ,
  0.05187988, -0.00298055, -0.08671061,  0.00607808,  0.11726888,
  0.02189128,  0.02555847,  0.09033203,  0.17886353,  0.10844072,
  0.10624186,  0.06652832,  0.06982422,  0.02102661, -0.0699056 ,
  0.01693726, -0.08265432,  0.2199707 ,  0.06274414, -0.19523112,
  0.12554932,  0.06892904,  0.12934367,  0.21972656,  0.07495117,
 -0.0929362 , -0.13761394,  0.06593832,  0.03621419, -0.04368083,
 -0.01843262,  0.05630493,  0.05786133, -0.01005046, -0.04972076,
 -0.03026327, -0.02469889, -0.01918538,  0.01245117, -0.07993571,
 -0.1265157 , -0.00217692,  0.13208008, -0.01228841,  0.1065267 ,
 -0.10501099, -0.13842773, -0.08565267, -0.22045898,  0.21923828,
  0.06449381,  0.08446121,  0.09305318, -0.05288696,  0.06181844,
 -0.02705892,  0.01998901, -0.00170898, -0.07596842, -0.05981445,
  0.07824198,  0.1056722 , -0.13442993, -0.0413208 , -0.16796875,
 -0.16210938, -0.1394043 , -0.11419169, -0.07287598,  0.10921224],
      dtype=float32)
```

```
In [81]: X_train=np.vstack(df['vectore'].values)
```

```
In [83]: y=df["spam/ham"].values
```

```
In [91]: len(X_train) == len(y_train)
```

```
Out[91]: True
```

```
In [86]: from sklearn.model_selection import train_test_split

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [87]: # X_train=np.vstack(df['vectore'].values)

          len(y_train)
```

```
Out[87]: 4457
```

In [88]: `len(X_train)`

Out[88]: 4457

In [89]: `model= LogisticRegression()`

In [90]: `model.fit(X_train,y_train)`

Out[90]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In []:

```
In [152]: # def pad_vector(vec, target_dim=300):
#         if len(vec) < target_dim:
#             return np.pad(vec, (0, target_dim - len(vec)), 'constant') # Pad with zeros
#         return vec

# # Apply padding to all vectors in 'vettore' column
# df['vettore'] = df['vettore'].apply(lambda x: pad_vector(x, target_dim=300))

# # Now, stack the vectors
# X = np.vstack(df['vettore'].values)
```

In [153]: `df['vettore'] = df['vettore'].apply(lambda x: pad_vector(x, target_dim=300))`

In [154]: `X = np.vstack(df['vettore'].values)`

In [157]: `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`

In [158]: `model= LogisticRegression()`

In [159]: `model.fit(X_train,y_train)`

Out[159]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [163]: `Y_predict=model.predict(X_test)`

In [165]: `print("Classification Report:\n", classification_report(y_test, Y_predict))`
`print("Confusion Matrix:\n", confusion_matrix(y_test, Y_predict))`

```
Classification Report:
              precision    recall  f1-score   support

     0       0.97       0.99       0.98        966
     1       0.92       0.79       0.85        149

 accuracy          0.96          0.96        1115
 macro avg         0.94          0.89          0.91        1115
 weighted avg      0.96          0.96          0.96        1115

Confusion Matrix:
[[956  10]
 [ 32 117]]
```

In [168]: `accuracy = accuracy_score(y_test, Y_predict)`
`print(f"Accuracy: {accuracy:.2f}")`

Accuracy: 0.96

```
In [170]: from sklearn.metrics import precision_score, recall_score, f1_score

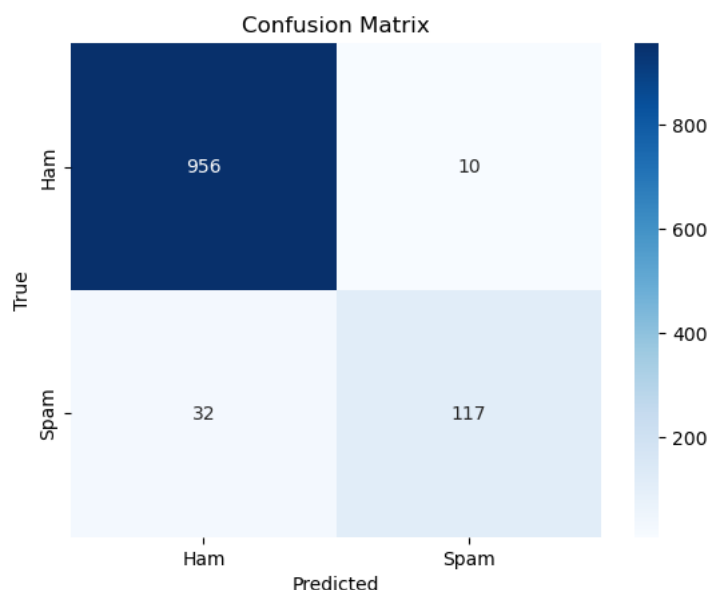
precision = precision_score(y_test, Y_predict)
recall = recall_score(y_test, Y_predict)
f1 = f1_score(y_test, Y_predict)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

Precision: 0.92
Recall: 0.79
F1-Score: 0.85

```
In [172]: from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, Y_predict)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In []:

```
In [146]: from nltk.tokenize import word_tokenize

# Example text
test_text = "cliam the prize the of one billion dolloar"

# Tokenize the text
tokens = word_tokenize(test_text.lower())

# Convert tokens to vectors using the Word2Vec model
test_vector = get_vectore(tokens) # Assuming `get_sentence_vector` is defined
```

```
In [147]: prdiction=model.predict([test_vector])
```

```
In [148]: prdiction
```

```
Out[148]: array([1], dtype=int64)
```

```
In [149]: if prdiction==1:
            print("spam")
        else:
            print("Ham")
```

spam

Train Our Own model

Feature	Bag of Words (BoW)	TF-IDF	Word2Vec
Based on word frequency	✓	✓ but weighted	✗
Captures word meaning	✗	✗	✓ (Semantic & syntactic)
Captures context	✗	✗	✓ (window-based context)
Sparse representation	✓	✓	✗ (dense vectors)
Use in deep learning	✗	✗	✓
Output format	Word count vector	Weighted vector	Dense word embeddings
Model type	Feature engineering	Feature engineering	Neural network (deep learning)

Project Sentiment Analysis (own Wed2vec model)

```
In [1]: import pandas as pd
import numpy as np
```

```
In [167]: df=pd.read_csv("amazon_reviews.csv")
```

```
In [168]: df1=df[["text", "overall"]]
```

```
In [169]: df1
```

```
Out[169]:
```

	text	overall
0	No issues.	4.0
1	Purchased this for my device, it worked as adv...	5.0
2	it works as expected. I should have sprung for...	4.0
3	This think has worked out great.Had a diff. br...	5.0
4	Bought it with Retail Packaging, arrived legit...	5.0
...	...	...
4910	I bought this Sandisk 16GB Class 10 to use wit...	1.0
4911	Used this for extending the capabilities of my...	5.0
4912	Great card that is very fast and reliable. It ...	5.0
4913	Good amount of space for the stuff I want to d...	5.0
4914	I've heard bad things about this 64gb Micro SD...	5.0

4915 rows × 2 columns

```
In [370]: df1["sentiment"]=df1["overall"].apply(lambda x : "positive " if x>=5 else "negative")
```

C:\Users\vicky\AppData\Local\Temp\ipykernel_26000\1855277397.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
df1["sentiment"]=df1["overall"].apply(lambda x : "positive " if x>=5 else "negative")

```
In [371]: df1
```

```
Out[371]:
```

	text	overall	sentiment	sentiment_cleaned	values	word
0	No issues.	4.0	negative	positive	1	[issu]
1	Purchased this for my device, it worked as adv...	5.0	positive	positive	1	[purchas, devic, work, advertis, never, much, ...
2	it works as expected. I should have sprung for...	4.0	negative	positive	1	[work, expect, sprung, higher, capac, think, m...
3	This think has worked out great.Had a diff. br...	5.0	positive	positive	1	[think, work, great, diff, bran, gb, card, wen...
4	Bought it with Retail Packaging, arrived legit...	5.0	positive	positive	1	[bought, retail, packag, arriv, legit, orang, ...
...	...	...	...	...	...	...
4910	I bought this Sandisk 16GB Class 10 to use wit...	1.0	negative	negative	0	[bought, sandisk, gb, class, use, htc, inspir,...
4911	Used this for extending the capabilities of my...	5.0	positive	positive	1	[use, extend, capabl, samsung, galaxi, note, g...
4912	Great card that is very fast and reliable. It ...	5.0	positive	positive	1	[great, card, fast, reliabl, come, option, ada...
4913	Good amount of space for the stuff I want to d...	5.0	positive	positive	1	[good, amount, space, stuff, want, fit, gopro,...
4914	I've heard bad things about this 64gb Micro SD...	5.0	positive	positive	1	[heard, bad, thing, gb, micro, sd, card, crap,...

4915 rows × 6 columns

```
In [372]: df1["sentiment"].value_counts()
df1["sentiment"].astype(str)
```

```
Out[372]: 0      negative
1      positive
2      negative
3      positive
4      positive
...
4910    negative
4911    positive
4912    positive
4913    positive
4914    positive
Name: sentiment, Length: 4915, dtype: object
```

```
In [373]: df1["sentiment_cleaned"] = df1["sentiment"].str.lower().str.strip()
```

C:\Users\vicky\AppData\Local\Temp\ipykernel_26000\3160913977.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
df1["sentiment_cleaned"] = df1["sentiment"].str.lower().str.strip()

```
In [374]: df1
```

```
Out[374]:
```

	text	overall	sentiment	sentiment_cleaned	values	word
0	No issues.	4.0	negative	negative	1	[issu]
1	Purchased this for my device, it worked as adv...	5.0	positive	positive	1	[purchas, devic, work, advertis, never, much, ...
2	it works as expected. I should have sprung for...	4.0	negative	negative	1	[work, expect, sprung, higher, capac, think, m...
3	This think has worked out great.Had a diff. br...	5.0	positive	positive	1	[think, work, great, diff, bran, gb, card, wen...
4	Bought it with Retail Packaging, arrived legit...	5.0	positive	positive	1	[bought, retail, packag, arriv, legit, orang, ...
...	...	...	...	...	...	...
4910	I bought this Sandisk 16GB Class 10 to use wit...	1.0	negative	negative	0	[bought, sandisk, gb, class, use, htc, inspir,...
4911	Used this for extending the capabilities of my...	5.0	positive	positive	1	[use, extend, capabl, samsung, galaxi, note, g...
4912	Great card that is very fast and reliable. It ...	5.0	positive	positive	1	[great, card, fast, reliabl, come, option, ada...
4913	Good amount of space for the stuff I want to d...	5.0	positive	positive	1	[good, amount, space, stuff, want, fit, gopro,...
4914	I've heard bad things about this 64gb Micro SD...	5.0	positive	positive	1	[heard, bad, thing, gb, micro, sd, card, crap,...

4915 rows × 6 columns

```
In [375]: # df["values"]=df["sentiment_cleaned"].apply(lambda x : 1 if x=="positive" else 0)
df1["values"] = df1["sentiment_cleaned"].apply(lambda x: 1 if x == "positive" else 0)
```

C:\Users\vicky\AppData\Local\Temp\ipykernel_26000\2530448986.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
df1["values"] = df1["sentiment_cleaned"].apply(lambda x: 1 if x == "positive" else 0)

```
In [376]: df1["sentiment_cleaned"].value_counts()
```

```
Out[376]: sentiment_cleaned
positive    3922
negative     993
Name: count, dtype: int64
```

```
In [377]: df1
```

```
Out[377]:
```

	text	overall	sentiment	sentiment_cleaned	values	word
0	No issues.	4.0	negative	negative	0	[issu]
1	Purchased this for my device, it worked as adv...	5.0	positive	positive	1	[purchas, devic, work, advertis, never, much, ...
2	it works as expected. I should have sprung for...	4.0	negative	negative	0	[work, expect, sprung, higher, capac, think, m...
3	This think has worked out great.Had a diff. br...	5.0	positive	positive	1	[think, work, great, diff, bran, gb, card, wen...
4	Bought it with Retail Packaging, arrived legit...	5.0	positive	positive	1	[bought, retail, packag, arriv, legit, orang, ...
...	...	...	...	...	...	...
4910	I bought this Sandisk 16GB Class 10 to use wit...	1.0	negative	negative	0	[bought, sandisk, gb, class, use, htc, inspir,...
4911	Used this for extending the capabilities of my...	5.0	positive	positive	1	[use, extend, capabl, samsung, galaxi, note, g...
4912	Great card that is very fast and reliable. It ...	5.0	positive	positive	1	[great, card, fast, reliabl, come, option, ada...
4913	Good amount of space for the stuff I want to d...	5.0	positive	positive	1	[good, amount, space, stuff, want, fit, gopro,...
4914	I've heard bad things about this 64gb Micro SD...	5.0	positive	positive	1	[heard, bad, thing, gb, micro, sd, card, crap,...

4915 rows × 6 columns

```
In [368]: df1["overall"].value_counts()
```

```
Out[368]: overall
5.0    3922
4.0     527
1.0     244
3.0     142
2.0      80
Name: count, dtype: int64
```

```
In [355]: df1["text"]=df1["text"].fillna("text")
df1["text"].isnull().sum()
```

C:\Users\vicky\AppData\Local\Temp\ipykernel_26000\323866282.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
df1["text"]=df1["text"].fillna("text")

```
Out[355]: 0
```

```
In [253]: y=df1["values"]
```

```
In [256]: import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
stemmer=PorterStemmer()
clean_text=[]
for i in range(0,len(df1)):
    review=re.sub("[^a-zA-Z0-9]", " ", df1["text"][i])
    review=review.lower()
    review=word_tokenize(review)
    review=[stemmer.stem(words) for words in review if words not in set(stopwords.words("english"))]
    review=" ".join(review)
    clean_text.append(review)
```

```
In [257]: clean_text
```

```
Out[257]: ['issu',
'purchas devic work advertis never much phone memori sinc download lot stuff brainer',
'work expect sprung higher capac think made bit cheesier earlier version paint look clean',
'think work great diff bran 64gb card went south 3 month one held pretti well sinc s3 note3 updat 3 21 14i month zero is
su sinc transfer s3 note3 note2 card reliabl solid cheer',
'bought retail packag arriv legit orang envelop english version asian like pictur show arriv quickli bought 32 16 retail
packag htc one sv lg optimu card work order probabl best price get nice sd card',
'mini storag anyth els suppos purchas add addit storag microsoft surfac pro tablet come 64 128 gb suppos sandisk long st
and reput speak',
'phone never skip beat file transfer speedi corrupt issu memori fade issu would expect sandisk brand great card entrust
preciou file slightli cheaper piec crap lose everyth forgiv spend extra coupl buck trust product goe good qa',
'hard believ afford digit becom 32 gb devic one quarter sie postag stamp would scienc fiction less gener ago pick portab
l music want schlep risk phone ipod work great sd card reader select confid',
'work htc rezound run short space 64gb sandisk order came fast issu',
'galaxi s4 super fast card total happi happi still type fill requir word though',
'like sd card take music video download person video file doc multimedia imag fast transfer rate class 10 speed take gam
e larg file easili still enough space app 34 great video camera camcord suppli adapt fit easili smartphon tablet sd card
slot recommend 32gb sd card everyon',
'work file write bit slower expect usb3 reader also read write faster card insid standard size sd adapt 15 mb vs 10 writ
```

```
In [259]: from gensim.utils import simple_preprocess
```

```
In [260]: words=list([simple_preprocess(i) for i in clean_text ])
```

```
In [261]: df1["word"]=words
```

C:\Users\vicky\AppData\Local\Temp\ipykernel_26000\3002329179.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
df1["word"]=words

```
In [262]: df1
```

Out[262]:

	text	overall	sentiment	sentiment_cleaned	values	word
0	No issues.	4.0	positive	positive	1	[issu]
1	Purchased this for my device, it worked as adv...	5.0	positive	positive	1	[purchas, devic, work, advertis, never, much, ...
2	it works as expected. I should have sprung for...	4.0	positive	positive	1	[work, expect, sprung, higher, capac, think, m...
3	This think has worked out great.Had a diff. br...	5.0	positive	positive	1	[think, work, great, diff, bran, gb, card, wen...
4	Bought it with Retail Packaging, arrived legit...	5.0	positive	positive	1	[bought, retail, packag, arriv, legit, orang, ...
...	...	...	...	...	...	...
4910	I bought this Sandisk 16GB Class 10 to use wit...	1.0	negative	negative	0	[bought, sandisk, gb, class, use, htc, inspir,...
4911	Used this for extending the capabilities of my...	5.0	positive	positive	1	[use, extend, capabl, samsung, galaxi, note, g...
4912	Great card that is very fast and reliable. It ...	5.0	positive	positive	1	[great, card, fast, reliabl, come, option, ada...
4913	Good amount of space for the stuff I want to d...	5.0	positive	positive	1	[good, amount, space, stuff, want, fit, gopro,...
4914	I've heard bad things about this 64gb Micro SD...	5.0	positive	positive	1	[heard, bad, thing, gb, micro, sd, card, crap,...

4915 rows × 6 columns

```
In [263]: from gensim.models import Word2Vec
```

```
In [ ]:
```

```
In [ ]:
```

Parameter	Meaning
words	Your input corpus, a list of tokenized sentences (list of lists of words)
vector_size	Dimensionality of the word embeddings (default used to be size , now vector_size)
window	Maximum distance between the current and predicted word within a sentence (context window)
min_count	Ignores all words with total frequency lower than this (helps ignore rare words/noise)
workers	Number of CPU cores to use for training (parallelization)
sg	Training algorithm: 1 = Skip-Gram, 0 = CBOW

```
In [ ]:
```

```
In [267]: model=Word2Vec(words,vector_size=100>window=5,min_count=1,workers=1,sg=1)
```

```
In [ ]:
```

```
In [268]: model.wv["work"].shape
```

Out[268]: (100,)

```
In [125]: # def get_vectore(tokens):  
#         return np.mean(model,axis=0) if model else np.zeros(vector_size)
```

```
In [269]: model.wv.index_to_key
```

Out[269]:

```
['card',  
'work',  
'use',  
'phone',  
'gb',  
'great',  
'memori',  
'sandisk',  
'sd',  
'one',  
'galaxi',  
'price',  
'speed',  
'fast',  
'good',  
'problem',  
'samsung',  
'bought',  
'storag',  
...]
```

```
In [134]: model.corpus_count
```

Out[134]: 4915

In [135]: `model.epochs`

Out[135]: 5

In [136]: `model.wv.most_similar("like",topn=10)`

Out[136]: `[('charm', 0.9250401854515076),
('wish', 0.9231306314468384),
('suppos', 0.922075629234314),
('kind', 0.9120535254478455),
('fact', 0.9074990153312683),
('guy', 0.9061111211776733),
('mind', 0.9041804075241089),
('afford', 0.903403103351593),
('rememb', 0.9019020199775696),
('inexpens', 0.901017427444458)]`

In [270]: `def avg_vector(doc):
 return np.mean([model.wv[word] for word in doc if word in model.wv.index_to_key],axis=0)`

In [271]: `import numpy as np
from tqdm import tqdm
x=[]

for i in tqdm(range(len(words))):
 x.append(avg_vector(words[i]))`

100%|██| 4915/4915 [00:02<00:00, 2204.58it/s]

In [272]: `x[0].shape`

Out[272]: (100,)

In [273]: `x=np.array(x)`

In [274]: `x.shape`

Out[274]: (4915, 100)

In [275]: `import pandas as pd

df = pd.DataFrame() # Create an empty DataFrame

Efficiently collect rows in a list
df_list = [pd.DataFrame(x[i].reshape(1, -1)) for i in range(len(x))]`

In [473]: `df_list[0]`

Out[473]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
0	-0.057833	-0.109053	-0.170496	-0.128066	0.243533	-0.440678	-0.100945	0.543786	-0.313419	-0.135994	...	0.294168	0.20466	-0.005035	0.083514

1 rows × 100 columns



In []:

In [276]: `df = pd.concat(df_list, ignore_index=True)

df.shape`

Out[276]: (4915, 100)

In [475]: df

Out[475]:

	0	1	2	3	4	5	6	7	8	9 ...	90	91	92	93
0	-0.057833	-0.109053	-0.170496	-0.128066	0.243533	-0.440678	-0.100945	0.543786	-0.313419	-0.135994 ...	0.294168	0.204660	-0.005035	0.083514
1	0.021938	0.050332	-0.281043	0.062285	0.110714	-0.320829	0.116434	0.547714	-0.231469	-0.198204 ...	0.344664	0.108889	0.024083	0.214785
2	0.051089	0.043672	-0.165627	0.010186	0.087131	-0.225849	0.113300	0.417951	-0.197151	-0.138087 ...	0.287096	0.079896	0.044970	0.075841
3	-0.004033	-0.039819	-0.221048	-0.078607	0.075877	-0.362453	0.049141	0.428240	-0.223341	-0.141554 ...	0.291284	0.130256	-0.019364	0.122975
4	0.017521	0.083249	-0.183332	0.007267	0.089473	-0.320578	0.130300	0.394428	-0.265965	-0.209586 ...	0.327425	0.116694	0.025624	0.077097
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4910	0.042381	-0.007923	-0.190171	0.031105	0.034508	-0.288403	0.034989	0.389050	-0.226551	-0.105261 ...	0.296593	0.083528	-0.013552	0.092585
4911	-0.086023	-0.017988	-0.198482	-0.061683	0.176021	-0.374022	0.006137	0.524017	-0.246974	-0.228682 ...	0.388943	0.135632	0.054789	0.240653
4912	0.034638	0.115907	-0.268474	0.016263	0.124347	-0.242431	0.173931	0.451923	-0.329360	-0.123478 ...	0.329822	0.059239	0.042485	0.076660
4913	0.039915	0.239924	-0.376130	0.051628	0.143091	-0.330065	0.264125	0.589091	-0.338702	-0.180492 ...	0.397114	0.123258	-0.004234	0.255015
4914	0.014500	0.060792	-0.207409	0.001620	0.074873	-0.319305	0.169881	0.446378	-0.283678	-0.148541 ...	0.300820	0.088045	0.009312	0.104082

4915 rows × 100 columns

In [277]: x=df

In [278]: x.isnull().sum()

Out[278]:

0	0
1	0
2	0
3	0
4	0
...	...
95	0
96	0
97	0
98	0
99	0

Length: 100, dtype: int64

In [378]: y=df1["sentiment"]
y.value_counts()

Out[378]:

sentiment	
positive	3922
negative	993

Name: count, dtype: int64

In [280]: x

Out[280]:

	0	1	2	3	4	5	6	7	8	9 ...	90	91	92	93
0	-0.057833	-0.109053	-0.170496	-0.128066	0.243533	-0.440678	-0.100945	0.543786	-0.313419	-0.135994 ...	0.294168	0.204660	-0.005035	0.083514
1	0.021938	0.050332	-0.281043	0.062285	0.110714	-0.320829	0.116434	0.547714	-0.231469	-0.198204 ...	0.344664	0.108889	0.024083	0.214785
2	0.051089	0.043672	-0.165627	0.010186	0.087131	-0.225849	0.113300	0.417951	-0.197151	-0.138087 ...	0.287096	0.079896	0.044970	0.075841
3	-0.004033	-0.039819	-0.221048	-0.078607	0.075877	-0.362453	0.049141	0.428240	-0.223341	-0.141554 ...	0.291284	0.130256	-0.019364	0.122975
4	0.017521	0.083249	-0.183332	0.007267	0.089473	-0.320578	0.130300	0.394428	-0.265965	-0.209586 ...	0.327425	0.116694	0.025624	0.077097
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4910	0.042381	-0.007923	-0.190171	0.031105	0.034508	-0.288403	0.034989	0.389050	-0.226551	-0.105261 ...	0.296593	0.083528	-0.013552	0.092585
4911	-0.086023	-0.017988	-0.198482	-0.061683	0.176021	-0.374022	0.006137	0.524017	-0.246974	-0.228682 ...	0.388943	0.135632	0.054789	0.240653
4912	0.034638	0.115907	-0.268474	0.016263	0.124347	-0.242431	0.173931	0.451923	-0.329360	-0.123478 ...	0.329822	0.059239	0.042485	0.076660
4913	0.039915	0.239924	-0.376130	0.051628	0.143091	-0.330065	0.264125	0.589091	-0.338702	-0.180492 ...	0.397114	0.123258	-0.004234	0.255015
4914	0.014500	0.060792	-0.207409	0.001620	0.074873	-0.319305	0.169881	0.446378	-0.283678	-0.148541 ...	0.300820	0.088045	0.009312	0.104082

4915 rows × 100 columns

In [379]: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2) # split the data into train and test set

In [380]: from sklearn.ensemble import RandomForestClassifier
MLmodel=RandomForestClassifier() # create a model
MLmodel.fit(x_train, y_train) # fit the model into the training data

Out[380]: RandomForestClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [381]: y_pred=MLmodel.predict(x_test)
```

```
In [382]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
In [383]: print("Accuracy: ", accuracy_score(y_test, y_pred)) # accuracy of the model
```

```
Accuracy: 0.844354018311292
```

```
In [384]: print(classification_report(y_test, y_pred))
```

```

              precision    recall  f1-score   support

   negative       0.79      0.35      0.48        206
   positive       0.85      0.98      0.91        777

   accuracy              0.84        983
  macro avg       0.82      0.66      0.70        983
 weighted avg     0.84      0.84      0.82        983

```

```
In [362]: confusion_matrix(y_test, y_pred)
```

```
Out[362]: array([[ 39,  54],
                 [ 22, 868]], dtype=int64)
```

```
In [393]: from nltk.tokenize import word_tokenize

text="Good product"

# Tokenize the text
tokens = word_tokenize(text.lower())

# Convert tokens to vectors using the Word2Vec model
test_vector = avg_vector(tokens) # Assuming `get_sentence_vector` is defined
```

```
In [394]: test_vector.shape
```

```
Out[394]: (100,)
```

```
In [395]: predict=MLmodel.predict([test_vector])
```

```
In [396]: print(predict)
```

```
['positive ']
```

IMDB dataset (Sentimen Analysis)

```
In [3]: import pandas as pd
data=pd.read_csv("IMDB-Dataset.csv")
```

```
In [4]: data.head()
```

```
Out[4]:
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
In [5]: data["sentiment"].value_counts()
```

```
Out[5]: sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

```
In [6]: dataf=data[:2010]
```

```
In [7]: dataf["sentiment"].value_counts()
```

```
Out[7]: sentiment
positive    1012
negative     998
Name: count, dtype: int64
```

```
In [8]: dataf
```

		review	sentiment
0	One of the other reviewers has mentioned that ...		positive
1	A wonderful little production. The...		positive
2	I thought this was a wonderful way to spend ti...		positive
3	Basically there's a family where a little boy ...		negative
4	Petter Mattei's "Love in the Time of Money" is...		positive
...		...	...
2005	I happened to catch this supposed "horror" fli...		negative
2006	This movie is my all time favorite!!! You real...		positive
2007	waste of 1h45 this nasty little film is one to...		negative
2008	I simply can't get over how brilliant the pair...		positive
2009	Stuck in a hotel in Kuwait, I happily switched...		positive

2010 rows x 2 columns

In []:

```
In [81]: import re
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
stemmer=PorterStemmer()
```

```
In [10]: stopwords.words("english")
```

'can',
'couldn't',
"couldn't",
'd',
'did',
'didn't',
"didn't",
'do',
'does',
'doesn't',
"doesn't",
'doing',
'don't',
"don't",
'down',
'during',
'each',
'few',
'for',
'from',

```
In [82]: dataM=[]
from tqdm import tqdm
for i in tqdm(range(0,len(dataf))):
    review=re.sub("[^a-zA-Z]", "",dataf['review'][i])
    review=review.lower()
    review=review.split()
    review=[ lemmatizer.lemmatize(word) for word in review if word not in set(stopwords.words("english"))]
    review=" ".join(review)
    dataM.append(review)
```

[illegible]

```
In [83]: from gensim.utils import simple preprocess
```

```
Sentence=[simple_preprocess(i) for i in dataM]
```

```
In [84]: from gensim.models import Word2Vec
W2V=Word2Vec(Sentence,vector_size=100,window=5,min_count=1,workers=4,sg=1)
```

```
In [85]: dataM[2007]
```

```
Out[85]: 'waste h nasty little film one avoid like cheap badly plotted cross saw recent film kidnap writer wrote obvious soul try mo
ney twist obvious peril could escape obvious etc good thing discovered new actress worth watching peyton list watch shuttle
though many better nicer film watch rather make miserable think le world spend time wisely watch good film exercise cook ni
ce meal anything waste time rubbish like'
```

```
In [86]: len(W2V.wv.index_to_key)
```

Out[86]: 22030

```
In [23]: W2V.wv.most_similar("like", topn=10)
```

```
Out[23]: [('forward', 0.8733320236260605),
('cool', 0.8584656715393066),
('cheap', 0.8493321537971497),
('monster', 0.8419457077980042),
('creatur', 0.8343626856803894),
('lot', 0.832798182964325),
('scream', 0.8318025469779968),
('type', 0.831409215927124),
('suppos', 0.8292579054832458),
('commerci', 0.8280846476554871)]
```

```
In [135]: import numpy as np
def avg_vector(doc):
    vectors = [W2V.wv[word] for word in doc if word in W2V.wv.index_to_key]
    if not vectors:
        return np.zeros(W2V.vector_size)
    return np.mean(vectors, axis=0)
```

[illegible]

```
In [137]: x=np.array(x)
```

```
In [138]: import pandas as pd

df=pd.DataFrame()

df_list = [pd.DataFrame(x[i].reshape(1, -1)) for i in range(len(x))]

df=pd.concat(df_list, ignore_index=True)
```

```
In [139]: X=df
```

```
In [140]: Y=dataf["sentiment"].map({"positive":1,"negative":0},)
```

In [141]: Y

```
Out[141]: 0      1
          1      1
          2      1
          3      0
          4      1
          ..
        2005      0
        2006      1
        2007      0
        2008      1
        2009      1
        Name: sentiment, Length: 2010, dtype: int64
```

```
In [142]: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) # split the data into train and test set
```

```
In [143]: from sklearn.ensemble import RandomForestClassifier
          ML=RandomForestClassifier() # create a model
          ML.fit(x_train, y_train) # fit the model into the training data
```

Out[143]:

RandomForestClassifier

RandomForestClassifier()

(<https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.RandomForestClassifier.html>)

```
In [144]: y_pred=ML.predict(x_test)
```

```
In [145]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
In [146]: print("Accuracy: ", accuracy_score(y_test, y_pred)) # accuracy of the model
```

Accuracy: 0.7437810945273632

```
In [147]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.74	0.77	0.76	208
1	0.74	0.72	0.73	194
accuracy			0.74	402
macro avg	0.74	0.74	0.74	402
weighted avg	0.74	0.74	0.74	402

```
In [176]: def clean(text):
review=re.sub('[^a-zA-Z]', " ", text)
review=review.lower()
review=review.split()
review=[lemmatizer.lemmatize(words) for words in review if not words in set(stopwords.words("english"))]
review=" ".join(review)
return review
```

```
In [195]: text="""
A breathtaking cinematic experience! The film had heart, humor, and depth, keeping me hooked from beginning to end. One of the
best movies I have ever seen.
"""
text=clean(text)
```

```
In [196]: text
```

```
Out[196]: 'breathtaking cinematic experience film heart humor depth keeping hooked beginning end one best movie seen long time'
```

```
In [197]: from nltk.tokenize import word_tokenize

# Tokenize the text
tokens = word_tokenize(text)

# Convert tokens to vectors using the Word2Vec model
test_vector = avg_vector(tokens) # Assuming `get_sentence_vector` is defined
```

```
In [198]: predict=ML.predict([test_vector])
```

```
In [199]: predict
```

```
Out[199]: array([1], dtype=int64)
```

```
In [200]: if predict==1:
print("Positive")
else:
print("Negative")
```

Positive

DeepLearning Coming Soon

```
In [ ]:
```