NLP
WITH PYTORCH

PART I

ABOUT
NATURAL LANGUAGE PROCESSING

ABOUT NLP

◆ 자연어 처리

- 사람들 의사소통에 사용되는 언어를 **자연어**
- 기계에서 자연어 **분석 & 이해 & 생성 & 처리**하는 **기술**을 **NLP**라함
- **기계에 인간의 언어를 이해 &인식한다는 점**에서 **AI 딥러닝의 한 분야**
- **1950년부터 기계 번역 기술 연구 시작**
- **1960년 말뭉치(Corpus) 데이터 활용 기계 학습 기반 통계적 자연어 처리**
- 최근 **Deep Learning기술로 기계 번역 & 자연어 생성**

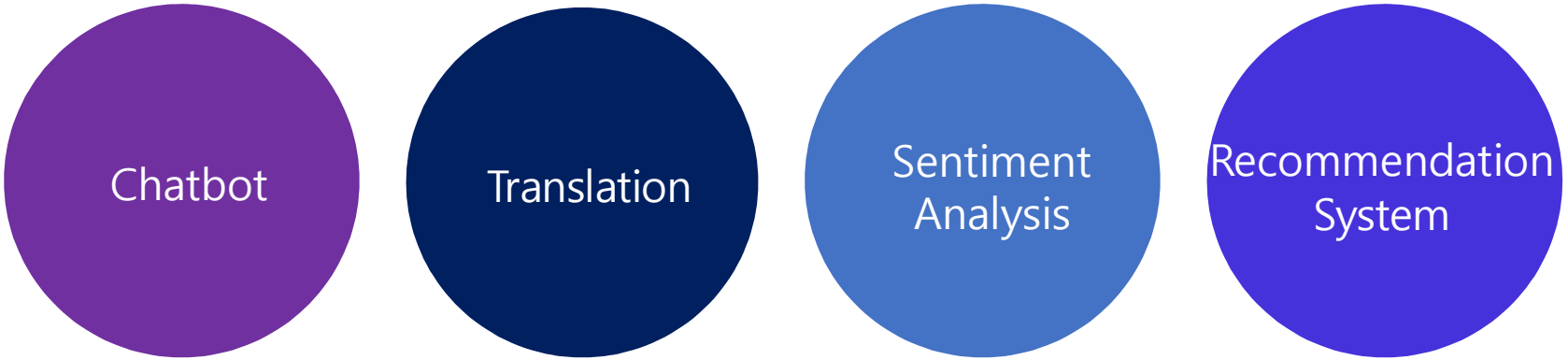
ABOUT NLP

◆ 자연어 처리 영역

• Machine Translation	하나의 언어를 다른 언어로 번역
• Sentiment Analysis	문장 감정상태 분석해서 긍정, 부정, N개 감정상태 분류
• Spam Filtering	텍스트의 스팸 여부 분류
• Image Captioning	이미지 설명 문장 생성
• Text Summarization	텍스트 요약문 자동 생성
• Question Answering	질문에 대한 정답 테스트 찾기
• Dialogue Generation	챗본 자동으로 텍스트 생성

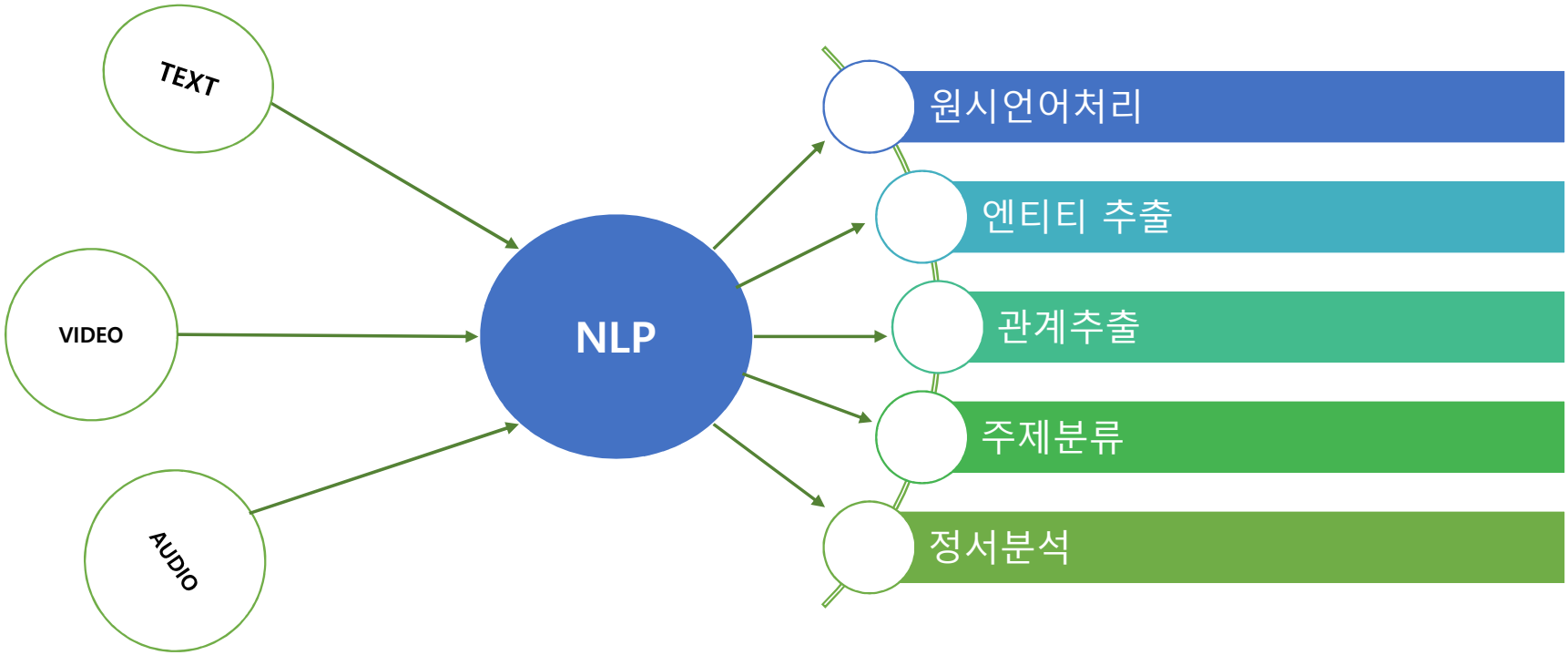
ABOUT NLP

◆ 자연어 처리 기반 AI 기술 응용 분야



ABOUT NLP

◆ 자연어 처리 기반 AI 기술 응용 분야



ABOUT NLP

◆ 자연어 처리 프로세스

ABOUT NLP

◆ 자연어 처리 프로세스 - 형태소 분석

- 자연 언어 **문장 구성의 최소 단위**인 **형태소** 단위 분활, 품사 판별
- 자연어 처리에서는 **토큰으로 형태소 이용**
- 분 석 : 어간 추출(stemming), 원형 복원(lemmatizing), 품사 부착(Part-Of-Speech tagging)
- 활 용 : 기계 번역, 텍스트 마이닝 분야

- 영어권 분석 방법 ➔ 띄어쓰기(공백) 기준 구분
- 아시아권 분석 방법 ➔ **문법 규칙** 방법
  ➔ **확률적 언어 모델** 방법

ABOUT NLP

◆ 자연어 처리 프로세스 - 형태소 분석

코퍼스 (Corpus)	• **말뭉치**라는 뜻으로, 자연어처리 위해 모아놓은 **텍스트 묶음** • 소설, 뉴스기사, 위키피디아 등에서 수집한 텍스트들
토 큰 (Token)	• **전체 문자열을 분석하고자 하는 단위**로 나눈 것 • 상황에 따라 **문장 단위, 단어 단위** 될 수도 있음
어휘 집합 (Vocabulary Set)	• **처리하는 문제영역의 전체 단어 집합** 의미 • 어휘 집합에 포함되지 않은 단어는 <UNK>(Unknown약) 특수 토큰 처리 • 충분히 큰 개수의 어휘집합 사용

ABOUT NLP

◆ 자연어 처리 프로세스 - 형태소 분석 패키지 준비

❖ NLTK 분석 패키지

[설치] conda install -c anaconda **nltk**

[설치 확인] import **nltk**

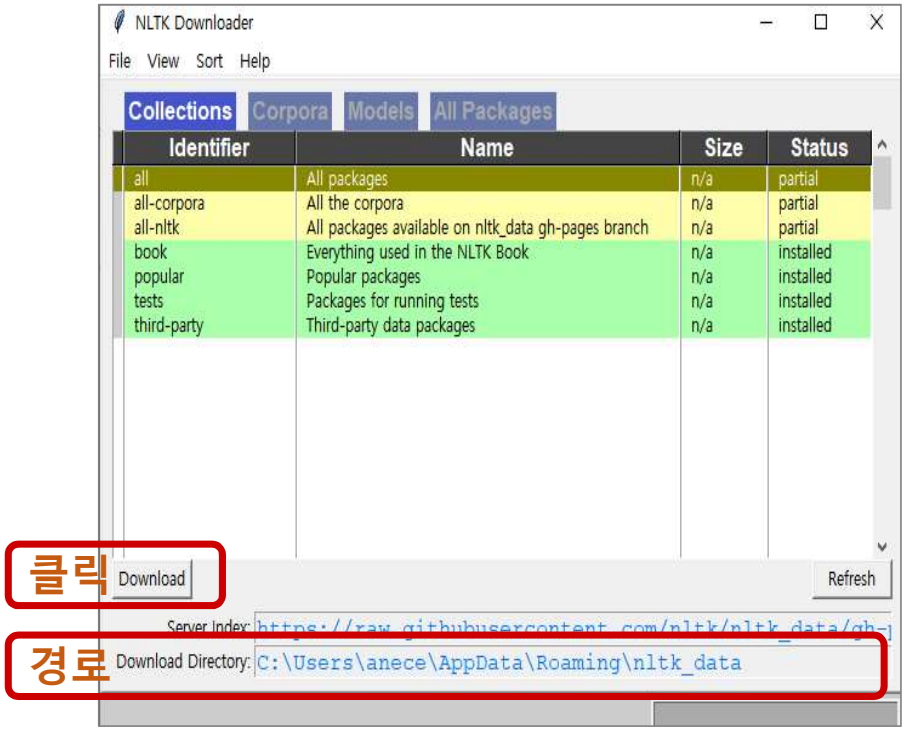
[데이터 세트 준비] **nltk.download( )** ← **말뭉치**

```
(TEXT_018_38) PS C:₩Users₩anece> python
Python 3.8.19 (default, Mar 20 2024, 19:55:45) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

ABOUT NLP

◆ 자연어 처리 프로세스 - 형태소 분석 패키지 준비

❖ NLTK 분석 패키지

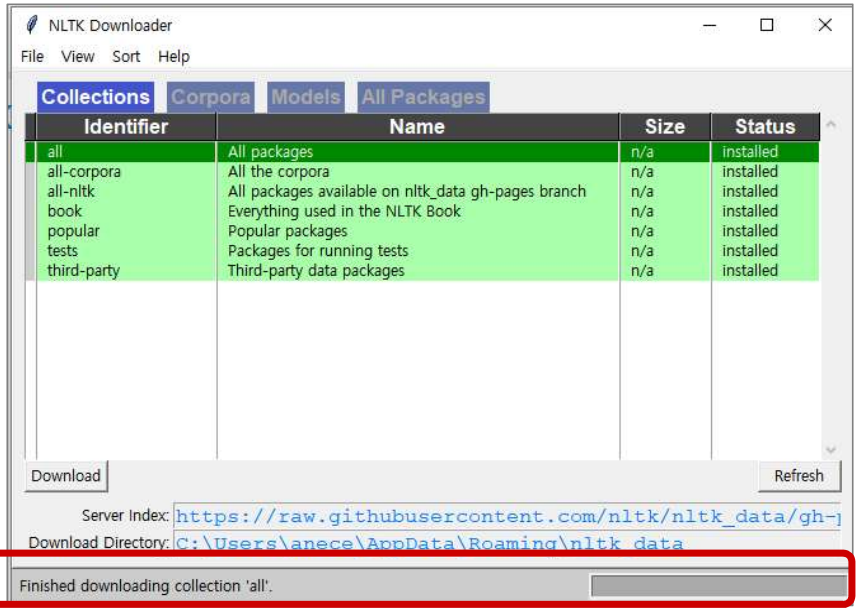


ABOUT NLP

◆ 자연어 처리 프로세스 - 형태소 분석 패키지 준비

❖ NLTK 분석 패키지

```
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
True
>>>
```



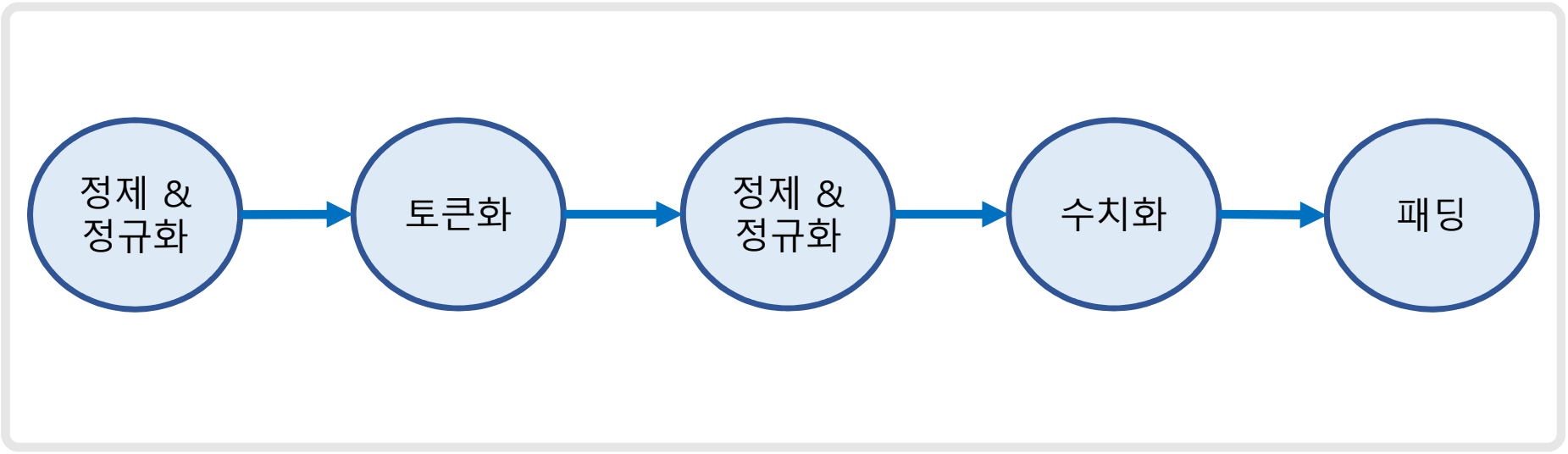
완료

PART I

TEXT PRERROCESSING

TEXT PREPROCESSING

◆ 전처리 단계



TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

❖ **말뭉치(corpus)에서 토큰(token)이라 불리는 단위로 나누는 작업**

❖ 의미있는 단위로 토큰 예로 **단어 토큰화, 문장 토근화**

❖ 정확한 토큰화를 위해 품사 활용 ➔ 품사 태깅 POS(Part-Of-Speech) Tagging

❖ **고려사항**

→구두점이나 특수 문자를 단순 제외해서는 안 됨 (예: m.p.h, Ph.D, AT&T , 2024-01-01, we're)

→줄임말과 단어 내에 띄어쓰기가 있는 경우 처리 방법

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

❖ **단어 토큰화 ➔ word_tokenize()**

```
from nltk.tokenize import word_tokenize


text="Happy, New Year! Don't stop. "


result = word_tokenize( text )
print(result)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

❖ **단어 토큰화 ➔ WordPunctTokenizer() : 구두점 분리 토큰화**

```
from nltk.tokenize import WordPunctTokenizer


text="Happy, New Year! Don't stop. "


wp_tokenizer = WordPunctTokenizer()
result = wp_tokenizer.tokenize(text)
print(result)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

❖ 문장 토큰화 ➔ sent_tokenize()

```
from nltk import sent_tokenize

text = 'The Matrix is everywhere its all around us, here even in this room. ₩
       You can see it out your window or on your television. ₩
       You feel it when you go to work, or go to church or pay your taxes.'

result = sent_tokenize( text )
print(result)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

```
from nltk.tokenize import word_tokenize, WordPunctTokenizer
from nltk.tokenize import TreebankWordTokenizer

msg="Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as
        cheery goes for a pastry shop."

# 단어 단위 토큰화 진행
print( f' [단어 토큰화 1] {word_tokenize(msg) }' )

# 구두점 분류 토큰화 진행
print( f' [단어 토큰화 2] { WordPunctTokenizer().tokenize(msg) }' )
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

```
# 표준 토큰화 진행
# - '하이픈(-)' 구성 단어는 하나로 유지
# - 어포스트로피로 구성 단어는 분리
print( f' [단어 토큰화 3] {TreebankWordTokenizer().tokenize(msg)} ')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

```
from nltk.tokenize import sent_tokenize

many_text = "His barber kept his word. But keeping such a huge secret to himself was driving ₩
            him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. ₩
            He dug a hole in the midst of some reeds. He looked about, to make sure no one ₩
            was near."

# 여러 개 문장으로 구성된 경우
print(f'[문장 토큰화1] {sent_tokenize(many_text)}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

```
# 한 개 문장으로 구성된 경우
one_text = "I am actively looking for Ph.D. students. and you are a Ph.D student."
print( f' [문장 토큰화2] {sent_tokenize(one_text)} ' )
```

TEXT PREPROCESSING

◆ 전처리 단계 - 토큰화(Tokenization)

```
from nltk.tag import pos_tag

text = "I am actively looking for Ph.D. students. and you are a Ph.D. student."
tokenized_sentence = word_tokenize(text)

# 일반 단어 토큰화
print(f'[단어 토큰화] f{tokenized_sentence}')

# 단어 품사 기반 토큰화
print(f'[품사 태  깅] f{pos_tag(tokenized_sentence)}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

❖ 표현 방법이 다른 단어 **통합하여 같은 단어 생성**

❖ **방법**

→ 대/소문자 통합

→ 같은 의미 다른 표기 단어들 통합 (예: USA , US) ➔ 어간 추출, 표제어 추출

→ 단위 통합

→ 의미론적 기반 단어 원형 추출

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

❖ 표제어 추출(Lemmatization)

→ **기본 사전형 단어를 표제어**라 함

→ 단어들이 다른 형태 가지더라도, 그 뿌리 단어 찾아서 단어의 개수 줄일 수 있는지 판단

→ 단어의 형태가 적절히 보존되는 양상을 보이는 특징

→ 방법 : 어간(stem)과 접사(affix) 분리

→ 예) am, are, is ---> be

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
# 표제어 추출 관련 모듈 로딩
from nltk.stem import WordNetLemmatizer

# 단어들
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly',
         'dies', 'watched', 'has', 'starting']

# 표제어 추출 인스턴스 생성
lemmatizer = WordNetLemmatizer()

# 표제어 추출 전 후 비교
print( '표제어 추출 전 :',words)
print( '표제어 추출 후 :',[lemmatizer.lemmatize(word) for word in words])
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
# 품사 정보 기반 표제어 추출
result=lemmatizer.lemmatize('dies', 'v')
print('동사 dies 표제어 추출 :',result)

result=lemmatizer.lemmatize('watched', 'v')
print('동사 watched 표제어 추출 :',result)

result=lemmatizer.lemmatize('has', 'v')
print('동사 has 표제어 추출 :',result)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

❖ 어간 추출(Stemming)

→ **형태학적 분석을 단순화한 버전**이라 함

→ 정해진 규칙만 보고 단어의 어미를 자르는 어림짐작 작업

→ 어간 추출 후에 나오는 결과 단어 → 사전에 존재하지 않는 단어일 수도 있음

→ 방법 : 어간(stem)과 접사(affix) 분리

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
# 어간 추출 모듈 로딩
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# 데이터
sentence = "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights and soundings--with the single exception of the red crosses and the written notes."
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
# 단어 토큰화  진행
tokenized_sentence = word_tokenize(sentence)
print('어간 추출 전 :', tokenized_sentence)

# 단어에서 어간 추출
stemmer = PorterStemmer()
print('어간 추출 후 :',[stemmer.stem(word) for word in tokenized_sentence])
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
# 알고리즘별 어간 추출 모듈 로딩
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer

# 어간 추출 인스턴스 생성
porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()

# 데이터
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has',
'starting']
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정규화(Normalization)

```
print('어간 추출 전 :', words)

print('포터 스테머의 어간 추출 후:',[porter_stemmer.stem(w) for w in words])

print('랭커스터 스테머의 어간 추출 후:',[lancaster_stemmer.stem(w) for w in words])
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

❖ **토큰화 작업 전/후** 토큰화 작업에 **방해가 되는 부분들 제거**

❖ 토큰화 작업 후 남아있는 노이즈들을 제거 위해 **지속적으로 진행**

❖ 방법 : 불용어 제거, 등장 빈도 기반 제거, 짧은 길이 단어 제거

❖ **완벽한 정제는 어려움**

❖ **노이즈 데이터(Noise Data)란?**

→ 분석 목적에 맞지 않는 단어들

→ 아무 의미도 갖지 않는 글자들

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

❖ 불용어 (Stopword)

- 의미가 없는 단어 토큰 제거하는 작업이 필요
- '큰 의미 없다'라는 것은 **자주 등장하지만 분석에 있어서는 큰 도움이 되지 않는 단어들**
- NLTK에서는 100여개 이상의 영어 단어들을 불용어로 패키지 내에서 미리 정의
- **개발자가 직접 정의**

  예) 조사, 접미사 같은 단어들

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

❖ **Stopwords 제거 => 불용어 패키지 다운로드**

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


stop_words_list = stopwords.words('english')


print('불용어 개수 :', len(stop_words_list))
print('불용어 10개 출력 :',stop_words_list[:10])
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

❖ **Stopwords 제거 => 불용어 처리**

```
example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for word in word_tokens:
    if word not in stop_words: result.append(word)

print(f'불용어 제거 전 : {word_tokens}, 불용어 제거 후 : {result}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

❖ 노이즈 데이터 특징 잡아서 제거 → 정규표현식

```
import re
text = "I was wondering if anyone out there could enlighten me on this car."

# 길이가 1~2인 단어들 정규 표현식 이용하여 삭제
shortword = re.compile(r'₩W*₩b₩w{1,2}₩b')
print(shortword.sub('', text))
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

```
# 불용어 및 토큰화 관련 모듈 로딩
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize


# NLTK의 영어 불용어 리스트 추출
stop_words_list = stopwords.words('english')
print(f'불용어 개수 : {len(stop_words_list)}개' )
print(f'불용어 10개 출력 : stop_words_list[:10]')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

```
# NLTK의 영어 불용어 추출
example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

# 문장 ==> 단어 토큰화
word_tokens = word_tokenize(example)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 정제(Cleaning)

```
# 불용어 제거
result = [ ]
for word in word_tokens:
    if word not in stop_words:  result.append(word)

print(f'불용어 제거 전 : {word_tokens}')
print(f'불용어 제거 후 : {result}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 벡터화/수치화 (Vectorization)

❖ **컴퓨터가 단어 인지를 위해 수치로 변환하는 것**

❖ 각 단어를 고유한 정수에 맵핑(mapping)

❖ 텍스트보다 숫자 연산이 더 빠름!

❖ 방식

→ **정수 인코딩** : 단어 빈도수 순 정렬된 단어 집합 생성, 빈도수 정수 부여 (경향성)

→ **원핫 인코딩** : 정수 인코딩 후 표현 단어 인덱스에 1부여, 원 핫 벡터 형성

→ **단어 분　리** : 단어 분리 후 의미 있는 단위로 나누어 해당 단어 이해

TEXT PREPROCESSING

◆ 전처리 단계 - 벡터화/수치화 (Vectorization)

```
# 단어 토큰화
vocab = {}
preprocessed_sentences = []
stop_words = set(stopwords.words('english'))

for sentence in sentences:
    # 단어 토큰화
    tokenized_sentence = word_tokenize(sentence)
    result = []

    for word in tokenized_sentence:
        word = word.lower()                 # 모든 단어를 소문자화하여 단어의 개수 줄임
        if word not in stop_words:          # 단어 토큰화 된 결과에 대해서 불용어를 제거
            if len(word) > 2:               # 단어 길이가 2이하인 경우에 대하여 추가로 단어 제거
                result.append(word)
                if word not in vocab:
                   vocab[word] = 0
                vocab[word] += 1
    preprocessed_sentences.append(result)

print(f'전처리 문장들\n{preprocessed_sentences}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 벡터화/수치화 (Vectorization)

```
print(f'단어 집합 : {vocab}')

# 'barber'라는 단어의 빈도수 출력
print(f'barber 단어 빈도수 : {vocab["barber"]}')

# 빈도수 순서로 정렬
vocab_sorted = sorted(vocab.items(), key = lambda x:x[1], reverse = True)
print(f'정렬된 단어사전 \n{vocab_sorted}')


# 높은 빈도수 가진 단어일수록 낮은 정수 부여
word_to_index = {}
i = 0
for (word, frequency) in vocab_sorted :
    if frequency > 1 :           # 빈도수가 작은 단어 제외
        i = i + 1
        word_to_index[word] = i

print(f'단어 정수 인덱싱\n{word_to_index}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 벡터화/수치화 (Vectorization)

```
# ----------------------------------------------------------
# 빈도수가 가장 높은 n개의 단어만 사용
# ----------------------------------------------------------
vocab_size = 5

# 인덱스가 5 초과인 단어 제거
words_frequency = [word for word, index in word_to_index.items() if index >= vocab_size + 1]

# 해당 단어에 대한 인덱스 정보를 삭제
for w in words_frequency:
    del word_to_index[w]
print(word_to_index)

# 단어 집합에 존재하지 않는 단어 ==> Out Of Vocab
word_to_index['OOV'] = len(word_to_index) + 1
print(word_to_index)
```

TEXT PREPROCESSING

◆ 전처리 단계 - 벡터화/수치화 (Vectorization)

```
# ----------------------------------------------------------
# 문장을 정수 인코딩
# ----------------------------------------------------------
encoded_sentences = []

for sentence in preprocessed_sentences:

    encoded_sentence = []
    for word in sentence:
        try:
            # 단어 집합에 있는 단어라면 해당 단어의 정수 리턴
            encoded_sentence.append(word_to_index[word])
        except KeyError:
            # 만약 단어 집합에 없는 단어라면 'OOV'의 정수 리턴.
            encoded_sentence.append(word_to_index['OOV'])
    encoded_sentences.append(encoded_sentence)

print(f'정수인코딩 문장\n{encoded_sentences}')
```

TEXT PREPROCESSING

◆ 전처리 단계 - 패딩(Padding)

❖ 각 문장 및 문서의 길이는 모두 다름

❖ **동일한 길이로 문장/문서를 맞추어 주는 작업**

❖ 동일 데이터에 대한 병렬 처리 위해 진행

TEXT PREPROCESSING

◆ 전처리 단계 - 패딩(Padding)

```
# ------------------------------------------------------------
# 길이가 다른 문장들 길이 통일
# ------------------------------------------------------------
max_len = max(len(item) for item in encoded_sentences)
print('최대 길이 :',max_len)

for sentence in encoded_sentences:
    while len(sentence) < max_len:
        sentence.append(0)

print(f'정수인코딩 - 패딩 후')
for sentence in encoded_sentences:
    print(sentence)
```