Codes:

- Python List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
fruits = ["apple","banana","cherry","kiwi","mango"]

newlist = [x for x in fruits if "a"in x]
```

```
df["word_tokenize"] = df['remove_puncuation'].apply(lambda x: _tokenize(x.lower()))
df.head()
```

- Reading dataset and Exploring dataset in Python

```
df = pd.read_csv("C:/Users/HP/Desktop/UQU/NLP/spam.csv",encoding = 'latin-1')
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
df = df.rename(columns={"v1":"label", "v2":"message"})
df.head()
```

	label	message
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...

```
pd.set_option('display.max_colwidth',100)
df.head()
```

	label	message
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...
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 around here though

```
df.isnull().value_counts()
```

```
label  message
False  False      5572
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   label    5572 non-null   object
 1   message  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
df.describe()
```

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

```
df.value_counts()
```

●Removing punctuation (Straing.punctuatuion)

Removing punctuation (Straing.punctuatuion)

```
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
text = "".join([word for word in text if word not in string.punctuation])
print(text)
```

```
DreamFactory Software is a Las Vegas Nevadabased software company DreamFactory develops both commercial and open source softwar
e that provides integrationplatformasaservice to multiple applications in cloudbased or on premise environments DreamFactory ma
y be deployed on premise or in the companys cloud environment
```

●Standardisation of document (Converting text to lowercase)

Standardisation of document (Converting text to lowercase)

```
print("".join([word.lower() for word in text]))
```

```
dreamfactory software is a las vegas nevadabased software company dreamfactory develops both commercial and open source softwar
e that provides integrationplatformasaservice to multiple applications in cloudbased or on premise environments dreamfactory ma
y be deployed on premise or in the companys cloud environment
```

●Tokenization in Python using :1- RegEx and split 2- NLTK

Tokenization in Python using :1- RegEx and split ¶

```
print(re.split(r"[.?!]",text))
```

```
['DreamFactory Software is a Las Vegas Nevadabased software company DreamFactory develops both commercial and open source softw
are that provides integrationplatformasaservice to multiple applications in cloudbased or on premise environments DreamFactory
may be deployed on premise or in the companys cloud environment']
```

```
print(text.split())
```

```
['DreamFactory', 'Software', 'is', 'a', 'Las', 'Vegas', 'Nevadabased', 'software', 'company', 'DreamFactory', 'develops', 'bot
h', 'commercial', 'and', 'open', 'source', 'software', 'that', 'provides', 'integrationplatformasaservice', 'to', 'multiple',
'applications', 'in', 'cloudbased', 'or', 'on', 'premise', 'environments', 'DreamFactory', 'may', 'be', 'deployed', 'on', 'prem
ise', 'or', 'in', 'the', 'companys', 'cloud', 'environment']
```

Tokenization in Python using :2- NLTK

```
tokenz = word_tokenize(text)
print(tokenz)
```

```
['DreamFactory', 'Software', 'is', 'a', 'Las', 'Vegas', 'Nevadabased', 'software', 'company', 'DreamFactory', 'develops', 'bot
h', 'commercial', 'and', 'open', 'source', 'software', 'that', 'provides', 'integrationplatformasaservice', 'to', 'multiple',
'applications', 'in', 'cloudbased', 'or', 'on', 'premise', 'environments', 'DreamFactory', 'may', 'be', 'deployed', 'on', 'prem
ise', 'or', 'in', 'the', 'companys', 'cloud', 'environment']
```

●Removing Stop- words using NLTK for bothe English and Arabic Languages

Removing Stop- words using NLTK for bothe English and Arabic Languages

```
stopwords = nltk.corpus.stopwords.words('english')
print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
print([word for word in tokenz if word not in stopwords])
```

```
['DreamFactory', 'Software', 'Las', 'Vegas', 'Nevadabased', 'software', 'company', 'DreamFactory', 'develops', 'commercial', 'o
pen', 'source', 'software', 'provides', 'integrationplatformasaservice', 'multiple', 'applications', 'cloudbased', 'premise',
'environments', 'DreamFactory', 'may', 'deployed', 'premise', 'companys', 'cloud', 'environment']
```

Most Common words

```
fdist = nltk.FreqDist(tokenz)
fdist.most_common()
```

```
[('DreamFactory', 3),
 ('software', 2),
 ('in', 2),
 ('or', 2),
 ('on', 2),
 ('premise', 2),
 ('Software', 1),
 ('is', 1),
 ('a', 1),
```

●Stemming using NLTK

```
#Stemming Example :

#Import stemming library :
from nltk.stem import PorterStemmer

porter = PorterStemmer()

#Word-list for stemming :
word_list = ["Study","Studying","Studies","Studied"]

for w in word_list:
    print(porter.stem(w))
```

```
studi
studi
studi
studi
```

```
#Stemming Example :

#Import stemming library :
from nltk.stem import PorterStemmer

porter = PorterStemmer()

#Word-list for stemming :
word_list = ["studies","leaves","decreases","plays"]

for w in word_list:
    print(porter.stem(w))
```

```
studi
leav
decreas
play
```

```
#Stemming Example :

#Import stemming library :
from nltk.stem import SnowballStemmer

snowball = SnowballStemmer("english")

#Word-list for stemming :
word_list = ["Study","Studying","Studies","Studied"]

for w in word_list:
    print(snowball.stem(w))
```

```
studi
studi
studi
studi
```

●Lemmatization unsing NLTK

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize('studies'))
```

```
study
```

```
from nltk import WordNetLemmatizer

lemma = WordNetLemmatizer()
word_list = ["Study","Studying","Studies","Studied"]

for w in word_list:
    print(lemma.lemmatize(w ,pos="v"))
```

```
Study
Studying
Studies
Studied
```

```
from nltk import WordNetLemmatizer

lemma = WordNetLemmatizer()
word_list = ["studies","leaves","decreases","plays"]

for w in word_list:
    print(lemma.lemmatize(w))
```

```
study
leaf
decrease
play
```

●N-grams and python

N-grams and python

```
ngram = pd.Series(nltk.ngrams(tokenz,3))
print(ngram)
```

```
0                        (DreamFactory, Software, is)
1                             (Software, is, a)
2                                  (is, a, Las)
3                               (a, Las, Vegas)
4                          (Las, Vegas, Nevadabased)
5                       (Vegas, Nevadabased, software)
6                    (Nevadabased, software, company)
7                  (software, company, DreamFactory)
8                  (company, DreamFactory, develops)
9                     (DreamFactory, develops, both)
```

●Counting vectorization

●N-gram Vectorization

4.1.1: Count vectorization

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
features_cv = vectorizer.fit_transform(sentences)
print(features_cv.shape)
print('Sparse Matrix :\n', features_cv)
features_cv = pd.DataFrame(features_cv.toarray())
features_cv.columns = vectorizer.get_feature_names()
features_cv
```

4.1.2: Vectorizing Data: N-Grams

```
ngram_vect = CountVectorizer(ngram_range=(1,3))
features_ng = ngram_vect.fit_transform(sentences)
print(features_ng.shape)
print('Sparse Matrix :\n', features_ng)
features_ng = pd.DataFrame(features_ng.toarray())
features_ng.columns = ngram_vect.get_feature_names()
features_ng
```

4.2.2: Vectorizing Data: N-Grams

```
ngram_vect = CountVectorizer(ngram_range=(1,3))
features_ngram = ngram_vect.fit_transform(df['cleaned_text'])
print(features_ngram.shape)
print('Sparse Matrix :\n', features_ngram)
features_ngram = pd.DataFrame(features_ngram.toarray())
features_ngram.columns = ngram_vect.get_feature_names()
features_ngram
```

```
(5568, 71114)
```

POS tagger

```
#PoS tagging :
tag = nltk.pos_tag(["Studying","Study"])
print (tag)
```

```
[('Studying', 'VBG'), ('Study', 'NN')]
```

```
#PoS tagging example :

sentence = "A very beautiful young lady is walking on the beach"

#Tokenizing words :
tokenized_words = word_tokenize(sentence)

for words in tokenized_words:
    tagged_words = nltk.pos_tag(tokenized_words)

tagged_words
```

```
[('A', 'DT'),
 ('very', 'RB'),
 ('beautiful', 'JJ'),
 ('young', 'JJ'),
 ('lady', 'NN'),
 ('is', 'VBZ'),
 ('walking', 'VBG'),
 ('on', 'IN'),
 ('the', 'DT'),
 ('beach', 'NN')]
```

```
#Extracting Noun Phrase from text :

# ? - optional character
# * - 0 or more repetations
grammar = "NP : {<DT>?<JJ>*<NN>} "

#Creating a parser :
parser = nltk.RegexpParser(grammar)

#Parsing text :
output = parser.parse(tagged_words)
print (output)

#To visualize :
output.draw()
```

```
(S
  A/DT
  very/RB
  (NP beautiful/JJ young/JJ lady/NN)
  is/VBZ
  walking/VBG
  on/IN
  (NP the/DT beach/NN))
```

```python
#Import required libraries :
from sklearn.feature_extraction.text import CountVectorizer

#Text for analysis :
sentences = ["Jim and Pam travelled by the bus:",
             "The train was late",
             "The flight was full.Travelling by flight is expensive"]

#Create an object :
cv = CountVectorizer()

#Generating output for Bag of Words :
B_O_W = cv.fit_transform(sentences).toarray()

#Total words with their index in model :
print(cv.vocabulary_)
print("\n")

#Features :
print(cv.get_feature_names())
print("\n")

#Show the output :
print(B_O_W)
```

```
{'jim': 7, 'and': 0, 'pam': 9, 'travelled': 12, 'by': 2, 'the': 10, 'bus': 1, 'train': 11, 'was': 14, 'late': 8, 'flight': 4,
'full': 5, 'travelling': 13, 'is': 6, 'expensive': 3}


['and', 'bus', 'by', 'expensive', 'flight', 'full', 'is', 'jim', 'late', 'pam', 'the', 'train', 'travelled', 'travelling', 'wa
s']


[[1 1 1 0 0 0 0 1 0 1 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 1 0 0 1]
 [0 0 1 1 2 1 1 0 0 0 1 0 0 1 1]]
```

```python
#Import required libraries :
from sklearn.feature_extraction.text import TfidfVectorizer

#Sentences for analysis :
sentences = ['This is the first document','This document is the second document']

#Create an object :
vectorizer = TfidfVectorizer(norm = None)

#Generating output for TF_IDF :
X = vectorizer.fit_transform(sentences).toarray()

#Total words with their index in model :
print(vectorizer.vocabulary_)
print("\n")

#Features :
print(vectorizer.get_feature_names())
print("\n")

#Show the output :
print(X)
```

```
{'this': 5, 'is': 2, 'the': 4, 'first': 1, 'document': 0, 'second': 3}


['document', 'first', 'is', 'second', 'the', 'this']


[[1.         1.40546511 1.         0.         1.         1.        ]
 [2.         0.         1.         1.40546511 1.         1.        ]]
```