Natural language processing

Practical list

1. To install Python Idle and NLTK.
2. To implement n grams model in NLP.
3. To implement Part of Speech tagging in NLP.
4. To implement kneeser ney smoothing technique in NLP.
5. To implementation of TF-Idf algorithm using python.
6. To implementation of Bag of words using python.
7. Implement one hot embedding modeling algorithm in NLP.
8. To implement text summarization in NLP.
9. To implement CountVectorizer in NLP.
10. To implement Skip gram model in NLP.

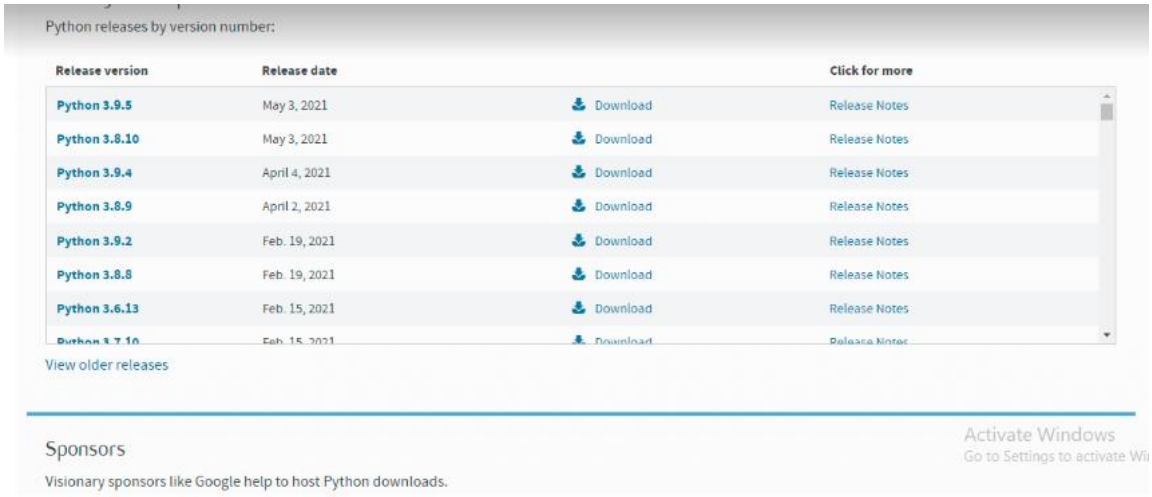
Practical 1

❖ **Aim**: To install Python Idle and NLTK.

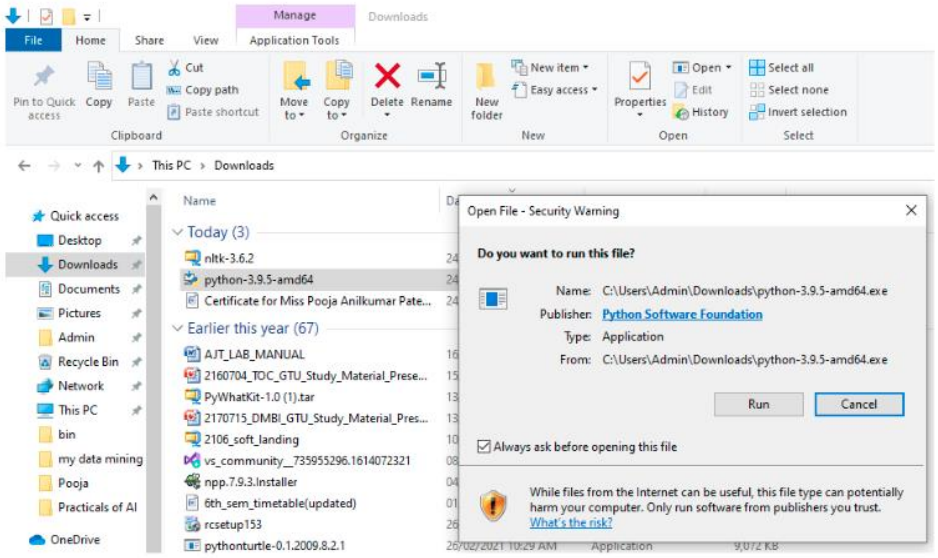
Step1: visit the link and download Python.

https://www.python.org/downloads/

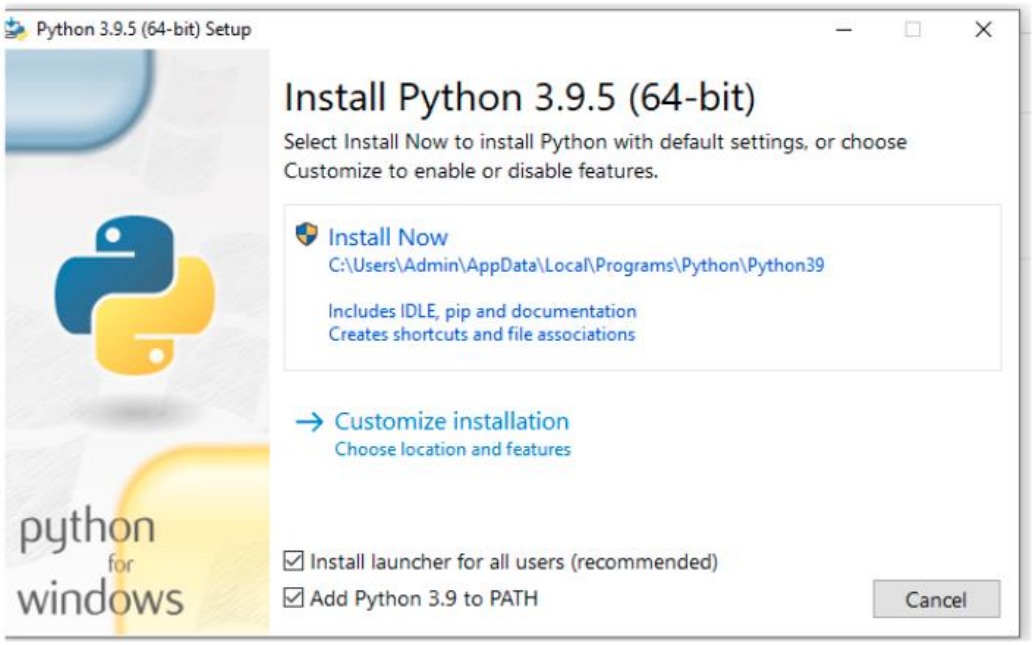
Step2: choose the latest version of Python .



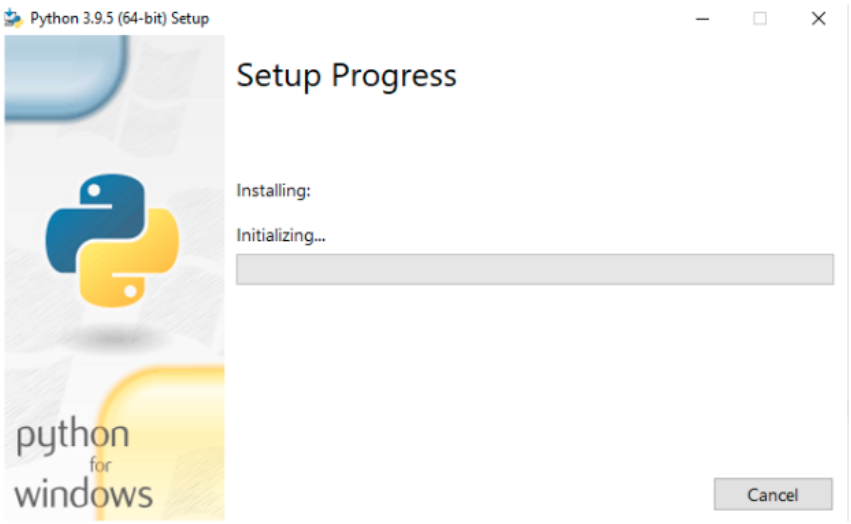
Step 3: select run.

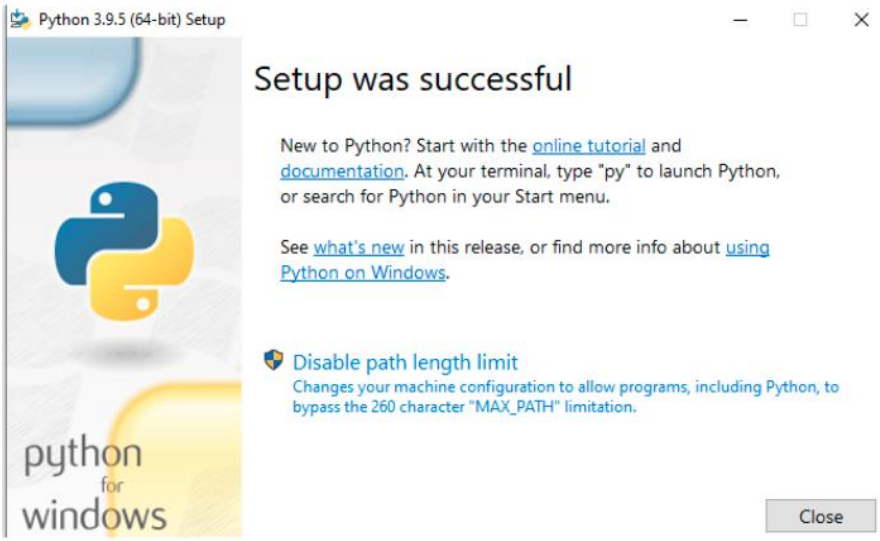


Step 4: select “Install now.”

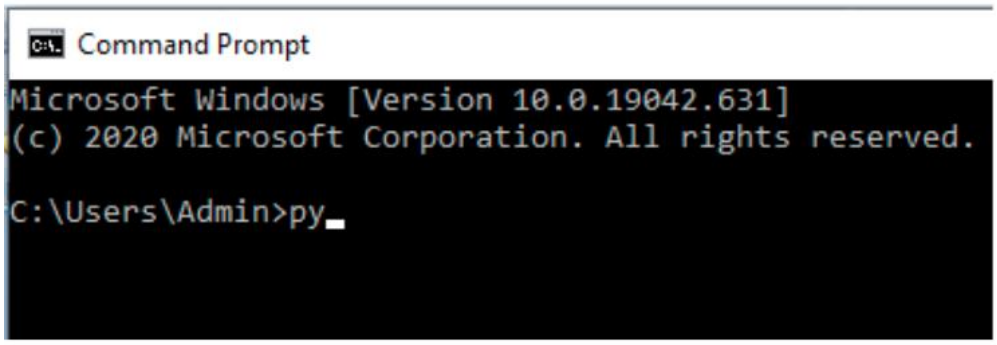


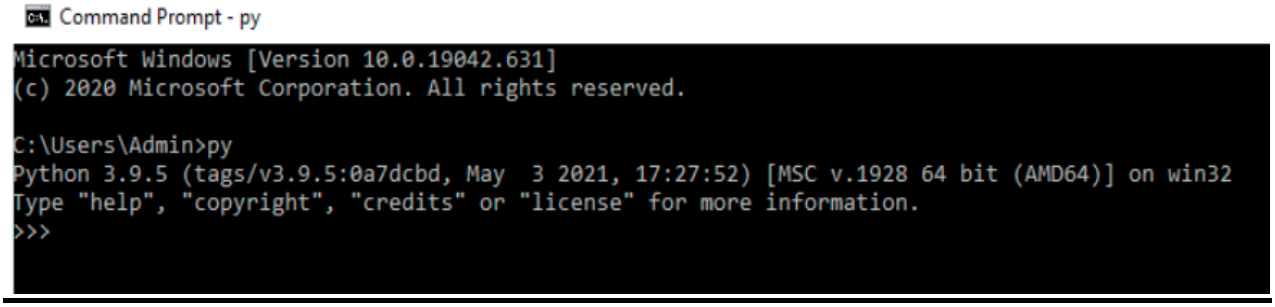
Step 5: then select “yes”.





Step 6: then confirm the python Version in command prompt by typing "py".

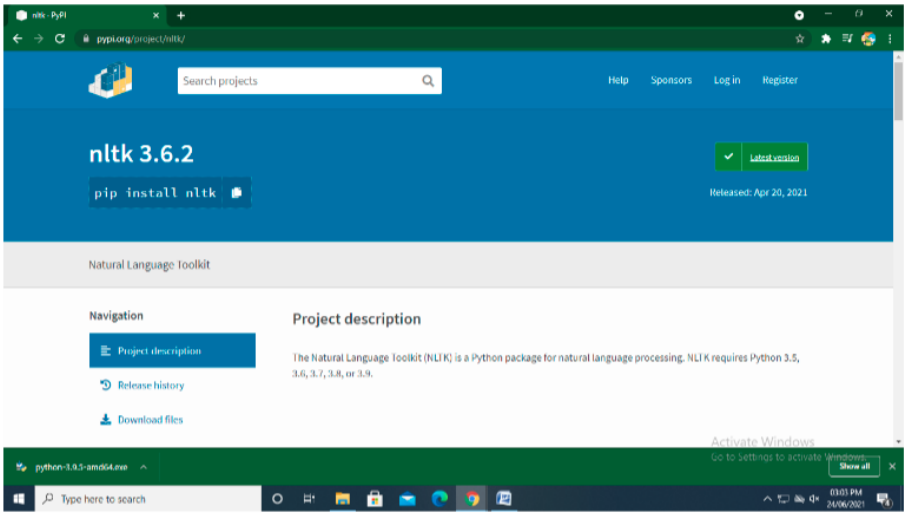




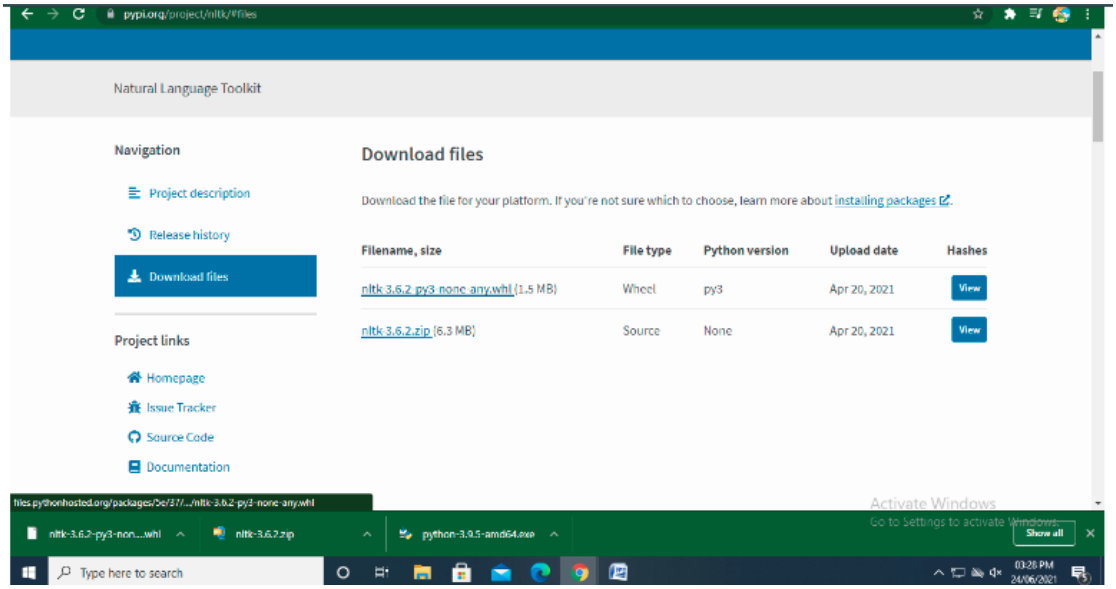
NLTK installation:

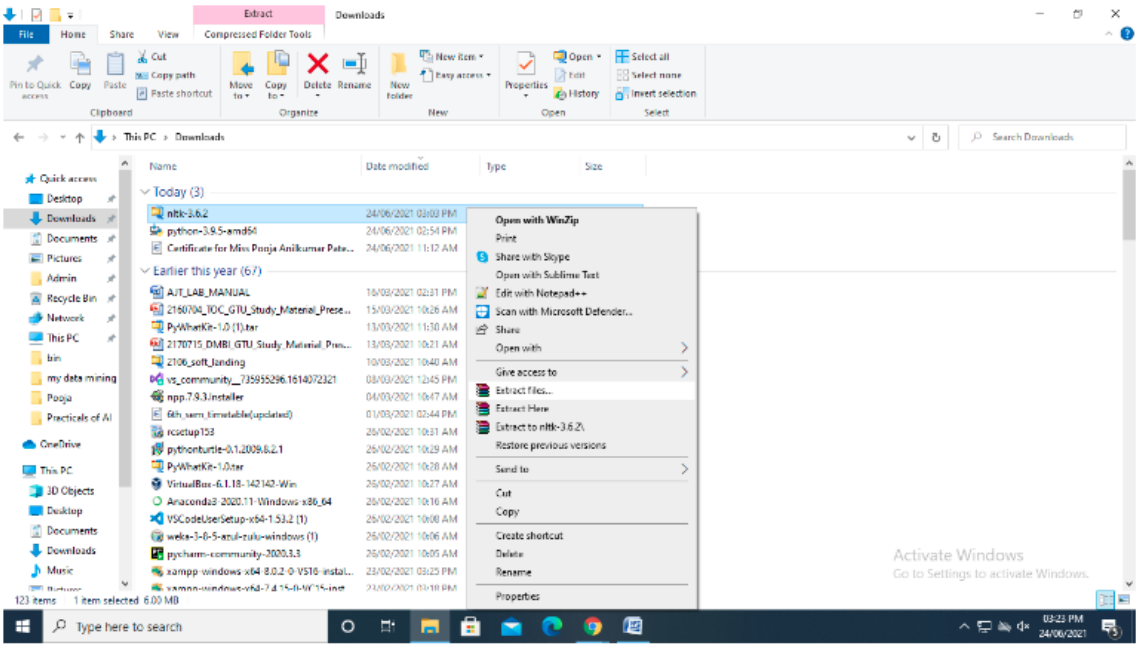
Step 1:Visit the link

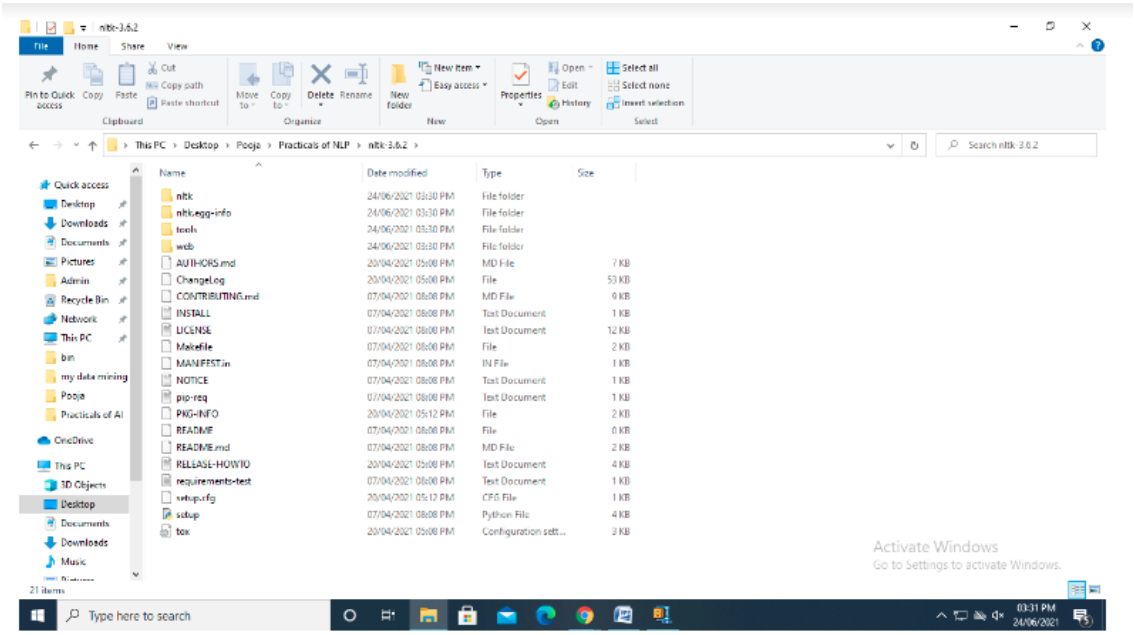
https://pypi.org/project/nltk/#files



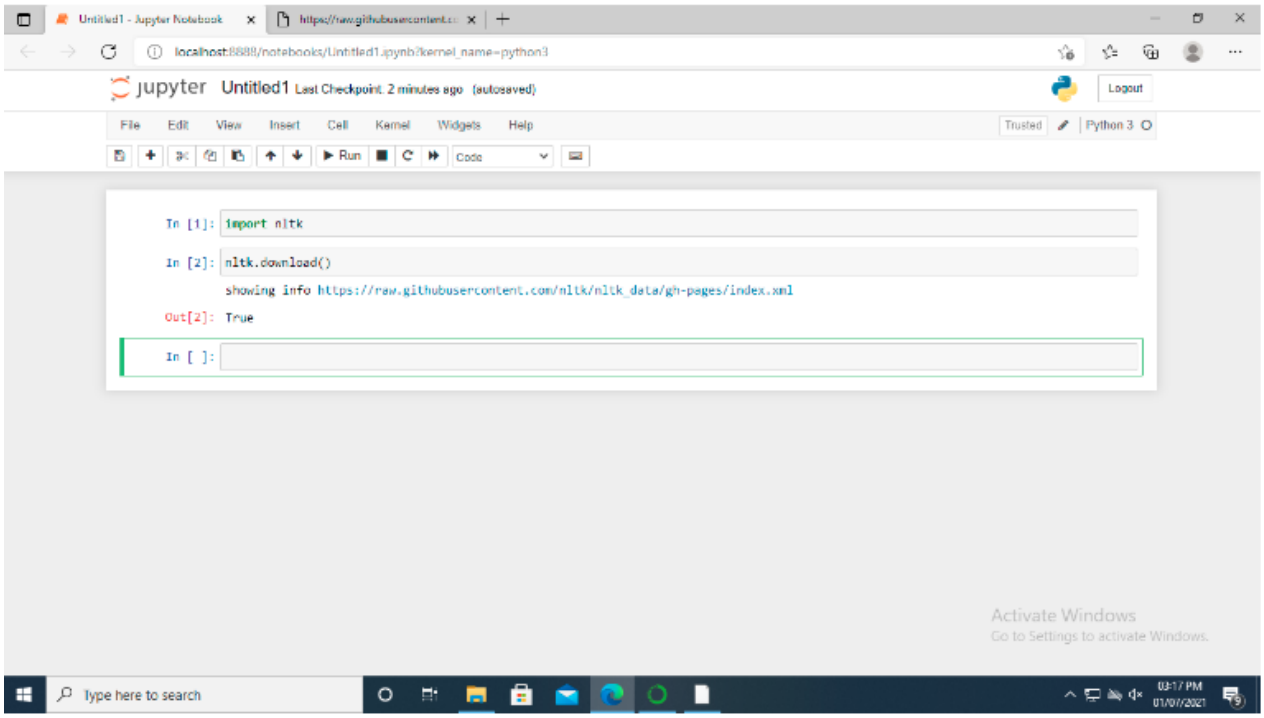
Step2: select download files and then extract the zip files.

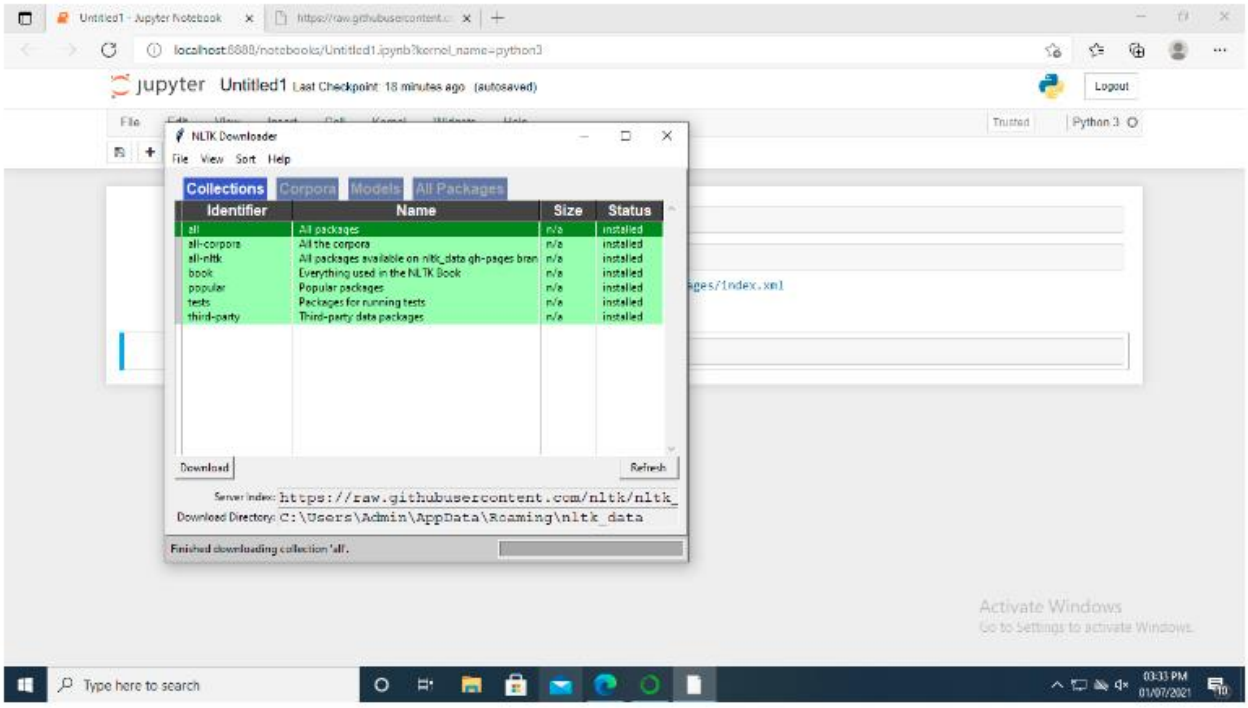






Step 3: open jupyter notebook and then download nltk packages.





Your natural language toolkit is installed!!!

Practical 2

❖ **Aim:** To implement n grams model in NLP.

Program:

```
#Implementation of unigram bigram and tigrsm in  NLP

#imports

import string

import random

import nltk

from nltk.util import pad_sequence

from nltk.util import bigrams

from nltk.util import ngrams

from nltk.util import everygrams

from nltk.lm.preprocessing import pad_both_ends

from nltk.lm.preprocessing import flatten

text = [['a', 'b', 'c'], ['a', 'c', 'd', 'c', 'e', 'f']]


list(bigrams(text[0]))  #bigrams
```

Output:

```
list(bigrams(text[0]))  #bigrams
```

```
Out[4]: [('a', 'b'), ('b', 'c')]
```

Out[4]: [('a', 'b'), ('b', 'c')]

```
In [5]: list(ngrams(text[1], n=3))  #tri-grams
Out[5]: [('a', 'c', 'd'), ('c', 'd', 'c'), ('d', 'c', 'e'), ('c', 'e', 'f')]

In [7]: list(ngrams(text[1], n=1))  #unigram
Out[7]: [('a',), ('c',), ('d',), ('c',), ('e',), ('f',)]

In [8]: list(ngrams(text[1], n=2))  #bigams
Out[8]: [('a', 'c'), ('c', 'd'), ('d', 'c'), ('c', 'e'), ('e', 'f')]

In [9]: list(ngrams(text[1], n=4))  #quadragrams
Out[9]: [('a', 'c', 'd', 'c'), ('c', 'd', 'c', 'e'), ('d', 'c', 'e', 'f')]

In [10]: list(ngrams(text[1], n=5)) #pentagrams
Out[10]: [('a', 'c', 'd', 'c', 'e'), ('c', 'd', 'c', 'e', 'f')]
```

Practical 3

❖ **Aim:** To implement Part of Speech tagging in NLP.

Program:

```
#implementation of POS tagging in NLP
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words(&#39;english&#39;))

# Dummy text
txt = &quot;Sukanya, Rajib and Naba are my good friends. &quot; \
&quot;Sukanya is getting married next year. &quot; \
&quot;Marriage is a big step in one’s life.&quot; \
&quot;It is both exciting and frightening. &quot; \
&quot;But friendship is a sacred bond between people.&quot; \
&quot;It is a special kind of love between us. &quot; \
&quot;Many of you must have tried searching for a friend &quot;\
&quot;but never found the right one.&quot;

# sent_tokenize is one of instances of
```

```
# PunktSentenceTokenizer from the nltk.tokenize.punkt module
tokenized = sent_tokenize(txt)
for i in tokenized:
# Word tokenizers is used to find the words
# and punctuation in a string
wordsList = nltk.word_tokenize(i)
# removing stop words from wordList
wordsList = [w for w in wordsList if not w in stop_words]
# Using a Tagger. Which is part-of-speech
# tagger or POS-tagger.
tagged = nltk.pos_tag(wordsList)
print(tagged)
```

output:

```
[('Sukanya', 'NNP'), (',', ','), ('Rajib', 'NNP'), ('Naba', 'NNP'), ('good', 'JJ'), ('friends', 'NNS'), ('.', '.')]
[('Sukanya', 'NNP'), ('getting', 'VBG'), ('married', 'VBN'), ('next', 'JJ'), ('year', 'NN'), ('.', '.')]
[('Marriage', 'NN'), ('big', 'JJ'), ('step', 'NN'), ('one', 'CD'), ('’', 'NN'), ('life.It', 'NN'), ('exciting', 'VBG'), ('frigh
tening', 'NN'), ('.', '.')]
[('But', 'CC'), ('friendship', 'NN'), ('sacred', 'VBD'), ('bond', 'NN'), ('people.It', 'NN'), ('special', 'JJ'), ('kind', 'N
N'), ('love', 'VB'), ('us', 'PRP'), ('.', '.')]
[('Many', 'JJ'), ('must', 'MD'), ('tried', 'VB'), ('searching', 'VBG'), ('friend', 'NN'), ('never', 'RB'), ('found', 'VBD'),
('right', 'JJ'), ('one', 'CD'), ('.', '.')]
```

Practical 4

❖ **Aim:** To implement kneeser ney smoothing technique in NLP.

Program:

```
#trigrams probability

#kneeser ney implementation

import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \

form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \

the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)

kneser_ney = nltk.KneserNeyProbDist(freq_dist)

prob_sum = 0

for i in kneser_ney.samples():

    prob_sum += kneser_ney.prob(i)

print(prob_sum)
```

Output:

```
40.881547619047 62
```

Example 1:

In [10]:
```
#trigrams probability
#kneeser ney implementation
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 0
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
40.88154761904762
```

Example 2:

In [33]:
```
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 1
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
41.88154761904762
```

Example 3:

In [34]:
```
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 0.5
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
41.38154761904762
```

Example 4:

In [35]:
```
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 0.16
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
41.04154761904762
```

Example 5:

In [38]:
```
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 2
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
42.881547619047616
```

In [39]:
```
import nltk

ngrams = nltk.trigrams("What a piece of work is man! how noble in reason! how infinite in faculty! in \
form and moving how express and admirable! in action how like an angel! in apprehension how like a god! \
the beauty of the world, the paragon of animals!")

freq_dist = nltk.FreqDist(ngrams)
kneser_ney = nltk.KneserNeyProbDist(freq_dist)
prob_sum = 3
for i in kneser_ney.samples():
    prob_sum += kneser_ney.prob(i)
print(prob_sum)
```

```
43.88154761904761
```

Practical 5

- Aim: To implementation of TF-Idf algorithm using python.

Program:

```
# import and instantiate TfidfVectorizer (with the default
parameters)
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer()
vect
# use TreeankWordTokenizer
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
vect.set_params(tokenizer=tokenizer.tokenize)
# remove English stop words
vect.set_params(stop_words=&#39;english&#39;)

# include 1-grams and 2-grams
vect.set_params(ngram_range=(1, 2))
# ignore terms that appear in more than 50% of the documents
vect.set_params(max_df=0.5)
# only keep terms that appear in at least 2 documents
```

vect.set_params(min_df=2)

output:

```
In [ ]: # import and instantiate TfidfVectorizer (with the default parameters)
        from sklearn.feature_extraction.text import TfidfVectorizer
        vect = TfidfVectorizer()
        vect

Out[1]: TfidfVectorizer()
```

```
In [2]: # use TreebankWordTokenizer
        from nltk.tokenize import TreebankWordTokenizer
        tokenizer = TreebankWordTokenizer()
        vect.set_params(tokenizer=tokenizer.tokenize)

        # remove English stop words
        vect.set_params(stop_words='english')

        # include 1-grams and 2-grams
        vect.set_params(ngram_range=(1, 2))

        # ignore terms that appear in more than 50% of the documents
        vect.set_params(max_df=0.5)

        # only keep terms that appear in at least 2 documents
        vect.set_params(min_df=2)

Out[2]: TfidfVectorizer(max_df=0.5, min_df=2, ngram_range=(1, 2), stop_words='english',
                        tokenizer=<bound method TreebankWordTokenizer.tokenize of <nltk.tokenize.treebank.TreebankWordTokenizer object
        at 0x000001B3E606CC70>>)
```

Practical 6

- Aim: To implementation of Bag of words using python.

Program:

```
#implement of bag of wrods in python

def vectorize(tokens):

&#39;&#39;&#39; This function takes list of words in a sentence as input

and returns a vector of size of filtered_vocab.It puts 0 if the

word is not present in tokens and count of token if present.&#39;&#39;&#39;

vector=[]

for w in filtered_vocab:

vector.append(tokens.count(w))

return vector

def unique(sequence):

&#39;&#39;&#39;This functions returns a list in which the order remains

same and no item repeats.Using the set() function does not

preserve the original ordering,so i didnt use that instead&#39;&#39;&#39;

seen = set()

return [x for x in sequence if not (x in seen or seen.add(x))]
```

```
#create a list of stopwords.You can import stopwords from nltk too

stopwords=[&quot;to&quot;,&quot;is&quot;,&quot;a&quot;]

#list of special characters.You can use regular expressions too

special_char=[&quot;,&quot;,&quot;:&quot;,&quot;
&quot;,&quot;;&quot;,&quot;.&quot;,&quot;?&quot;]

#Write the sentences in the corpus,in our case, just two

string1=&quot;It was the best of times&quot;

string2=&quot;It was the worst of times&quot;

#convert them to lower case

string1=string1.lower()

string2=string2.lower()

#split the sentences into tokens

tokens1=string1.split()

tokens2=string2.split()

print(tokens1)

print(tokens2)

#create a vocabulary list

vocab=unique(tokens1+tokens2)

print(vocab)

#filter the vocabulary list

filtered_vocab=[]
```

```
for w in vocab:
if w not in stopwords and w not in special_char:
filtered_vocab.append(w)
print(filtered_vocab)
#convert sentences into vectords
vector1=vectorize(tokens1)
print(vector1)
vector2=vectorize(tokens2)
print(vector2)
```

output:

```
['it', 'was', 'the', 'best', 'of', 'times']
['it', 'was', 'the', 'worst', 'of', 'times']
['it', 'was', 'the', 'best', 'of', 'times', 'worst']
['it', 'was', 'the', 'best', 'of', 'times', 'worst']
[1, 1, 1, 1, 1, 1, 0]
[1, 1, 1, 0, 1, 1, 1]
```

Practical 7

❖ **Aim:** Implement one hot embedding modeling algorithm in NLP.

Program:

```
#one hot encoding


import numpy as np

samples = {&#39;Jupiter has 79 known moons .&#39;, &#39;Neptune has 14 confirmed

moons !&#39;} # Sample set for our example

# Create an empty dictionary

token_index = {}

#Create a counter for counting the number of key-value pairs in the

token_length

counter = 0


# Select the elements of the samples which are the two sentences

for sample in samples:

for considered_word in sample.split():

if considered_word not in token_index:


# If the considered word is not present in the dictionary token_index,
```

add it to the token_index

The index of the word in the dictionary begins from 1

NLP(2170723)Enrollment_no.:181010107008

token_index.update({considered_word : counter + 1})

updating the value of counter

counter = counter + 1

print(token_index)

Output:

```
[14]: print(token_index)
{'Neptune': 1, 'has': 2, '14': 3, 'confirmed': 4, 'moons': 5, '!': 6, 'Jupiter': 7, '79': 8, 'known': 9, '.': 10}
```

Practical 8

- **Aim**: To implement text summarization in NLP.

Program:

```
#To implement text summarization in python.

# importing libraries

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize, sent_tokenize

# Input text - to summarize

text = &amp;quot;&amp;quot;&amp;quot; There are many techniques available to generate

extractive summarization to keep it simple,

I will be using an unsupervised learning approach to find the

sentences similarity and rank them.

Summarization can be defined as a task of producing a concise and

fluent summary while preserving key

information and overall meaning. One benefit of this will be,

you don’t need to train and build a model prior start using it for

your project.

It’s good to understand Cosine similarity to make the best use of

the code you are going to see.
```

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product

space that measures the cosine of the angle between them. Its measures cosine of the angle between vectors.

The angle will be 0 if sentences are similar."""

Tokenizing the text

stopWords = set(stopwords.words("english"))

words = word_tokenize(text)

Creating a frequency table to keep the

score of each word

freqTable = dict()

for word in words:

word = word.lower()

if word in stopWords:

continue

if word in freqTable:

freqTable[word] += 1

else:

freqTable[word] = 1

Creating a dictionary to keep the score

```
# of each sentence
sentences = sent_tokenize(text)
sentenceValue = dict()
for sentence in sentences:
for word, freq in freqTable.items():
if word in sentence.lower():
if sentence in sentenceValue:
sentenceValue[sentence] += freq
else:
sentenceValue[sentence] = freq
sumValues = 1
for sentence in sentenceValue:
sumValues += sentenceValue[sentence]
# Average value of a sentence from the original text

average = int(sumValues/len(sentenceValue))
# Storing sentences into our summary.
summary = &#39;&#39;
for sentence in sentences:
if (sentence in sentenceValue) and (sentenceValue[sentence] &gt;
```

(1.2 * average)):

summary += " " + sentence

print(summary)

Output:

```
There are many techniques available to generate extractive summarization to keep it simple, I will be using an unsupervised l
earning approach to find the sentences similarity and rank them. Cosine similarity is a measure of similarity between two non-z
ero vectors of an inner product space that measures the cosine of the angle between them.
```

Practical 9

❖ **Aim**: To implement CountVectorizer in NLP.

Program:

```
from sklearn.feature_extraction.text import CountVectorizer


document = [&amp;quot;My name is Pooja&amp;quot;,
&amp;quot;my name is Aditya&amp;quot;,
&amp;quot;Each Friend helps many other friends at


anywhere&amp;quot;]


# Create a Vectorizer Object
vectorizer = CountVectorizer()
vectorizer.fit(document)


# Printing the identified Unique words along with their indices
print(&amp;quot;Vocabulary: &amp;quot;, vectorizer.vocabulary_)


# Encode the Document
```

vector = vectorizer.transform(document)

Summarizing the Encoded Texts

print("Encoded Document is:")

print(vector.toarray())

Output:

```
: from sklearn.feature_extraction.text import CountVectorizer

document = ["My name is Pooja",
           "my name is Aditya",
           "Each Friend helps many other friends at anywhere"]

# Create a Vectorizer Object
vectorizer = CountVectorizer()

vectorizer.fit(document)

# Printing the identified Unique words along with their indices
print("Vocabulary: ", vectorizer.vocabulary_)

# Encode the Document
vector = vectorizer.transform(document)

# Summarizing the Encoded Texts
print("Encoded Document is:")
print(vector.toarray())
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/sklearn/feature_extraction/image.py:167: DeprecationWarning: `np.int
` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify a
ny behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precis
ion. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  dtype=np.int):
```



```
Vocabulary:  {'my': 9, 'name': 10, 'is': 7, 'pooja': 12, 'aditya': 0, 'each': 3, 'friend': 4, 'helps': 6, 'many': 8, 'oth
er': 11, 'friends': 5, 'at': 2, 'anywhere': 1}
Encoded Document is:
[[0 0 0 0 0 0 0 1 0 1 1 0 1]
 [1 0 0 0 0 0 0 1 0 1 1 0 0]
 [0 1 1 1 1 1 1 0 1 0 0 1 0]]
```

Practical 10

❖ **Aim:** To implement Skip gram model in NLP.

Program:

"Note: this code is divided in 3 parts"

➢ Part 1:

```
#Skip gram model in python
#code part 1
import numpy as np
import string
from nltk.corpus import stopwords

def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

class word2vec(object):
    def __init__(self):
        self.N = 10
```

```
        self.X_train = []
        self.y_train = []
        self.window_size = 2
        self.alpha = 0.001
        self.words = []
        self.word_index = {}

    def initialize(self,V,data):
        self.V = V
        self.W = np.random.uniform(-0.8, 0.8, (self.V, self.N))
        self.W1 = np.random.uniform(-0.8, 0.8, (self.N, self.V))

        self.words = data
        for i in range(len(data)):
            self.word_index[data[i]] = i


    def feed_forward(self,X):
        self.h = np.dot(self.W.T,X).reshape(self.N,1)
        self.u = np.dot(self.W1.T,self.h)
```

```
        #print(self.u)
        self.y = softmax(self.u)
        return self.y

    def backpropagate(self,x,t):
        e = self.y - np.asarray(t).reshape(self.V,1)
        # e.shape is V x 1
        dLdW1 = np.dot(self.h,e.T)
        X = np.array(x).reshape(self.V,1)
        dLdW = np.dot(X, np.dot(self.W1,e).T)
        self.W1 = self.W1 - self.alpha*dLdW1
        self.W = self.W - self.alpha*dLdW

    def train(self,epochs):
        for x in range(1,epochs):
            self.loss = 0
            for j in range(len(self.X_train)):
                self.feed_forward(self.X_train[j])
                self.backpropagate(self.X_train[j],self.y_train[j])
                C = 0
```

```
                for m in range(self.V):
                    if(self.y_train[j][m]):
                        self.loss += -1*self.u[m][0]
                        C += 1
                self.loss += C*np.log(np.sum(np.exp(self.u)))
            print("epoch ",x, " loss = ",self.loss)
            self.alpha *= 1/( (1+self.alpha*x) )

    def predict(self,word,number_of_predictions):
        if word in self.words:
            index = self.word_index[word]
            X = [0 for i in range(self.V)]
            X[index] = 1
            prediction = self.feed_forward(X)
            output = {}
            for i in range(self.V):
                output[prediction[i][0]] = i

            top_context_words = []
            for k in sorted(output,reverse=True):
```

```
                    top_context_words.append(self.words[output[k]])

    if(len(top_context_words)>=number_of_predictions):
                        break


            return top_context_words
        else:
            print("Word not found in dicitonary")
```

- Part 2:

```
#Skip gram model in python
#code part 2

def preprocessing(corpus):
    stop_words = set(stopwords.words('english'))
    training_data = []
    sentences = corpus.split(".")
    for i in range(len(sentences)):
        sentences[i] = sentences[i].strip()
        sentence = sentences[i].split()
        x = [word.strip(string.punctuation) for word in sentence
                                                if word not in
stop_words]
        x = [word.lower() for word in x]
        training_data.append(x)
```

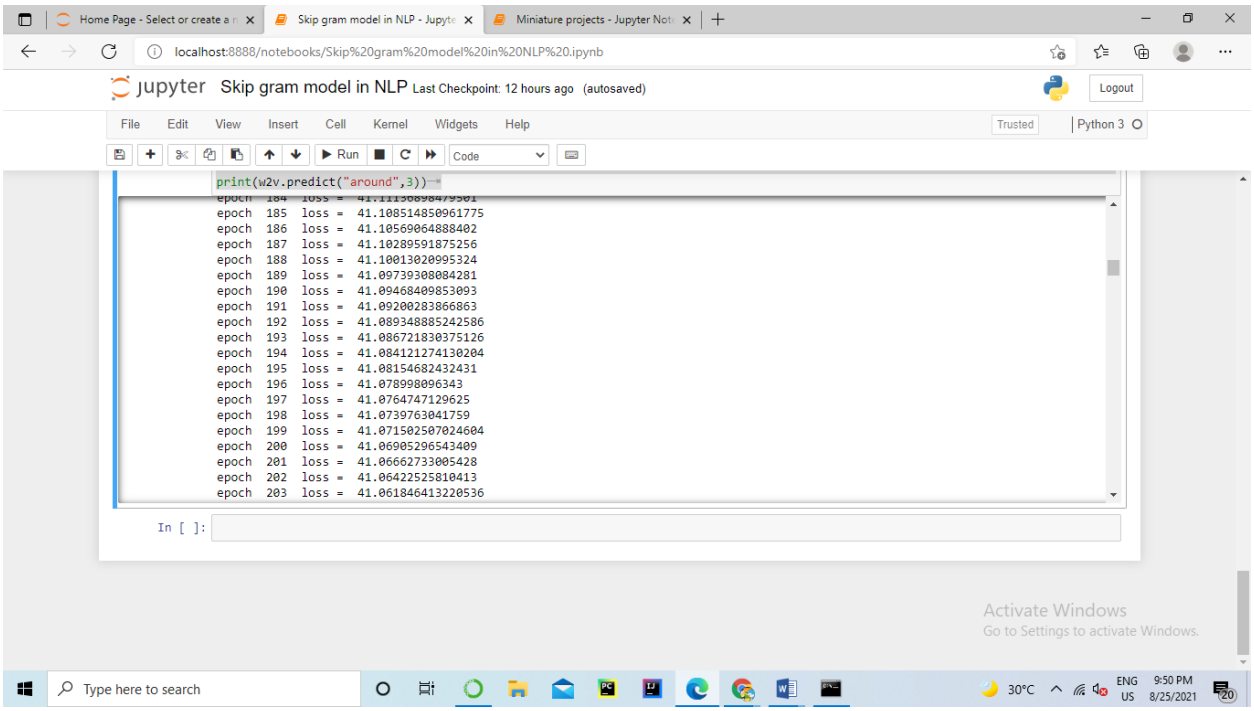
```
    return training_data


def prepare_data_for_training(sentences,w2v):
    data = {}
    for sentence in sentences:
        for word in sentence:
            if word not in data:
                data[word] = 1
            else:
                data[word] += 1
    V = len(data)
    data = sorted(list(data.keys()))
    vocab = {}
    for i in range(len(data)):
        vocab[data[i]] = i

    #for i in range(len(words)):
    for sentence in sentences:
        for i in range(len(sentence)):
            center_word = [0 for x in range(V)]
            center_word[vocab[sentence[i]]] = 1
            context = [0 for x in range(V)]

            for j in range(i-
w2v.window_size,i+w2v.window_size):
                if i!=j and j>=0 and j<len(sentence):
                    context[vocab[sentence[j]]] += 1
            w2v.X_train.append(center_word)
            w2v.y_train.append(context)
```

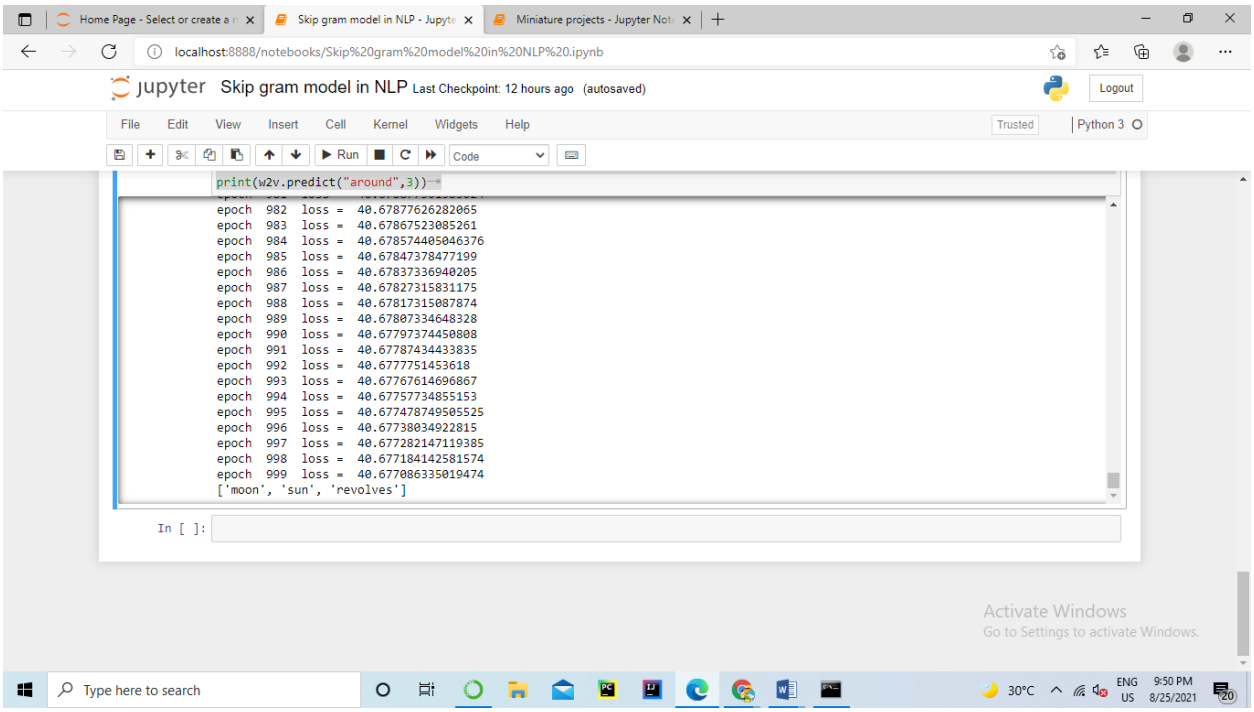
```
        w2v.initialize(V,data)

        return w2v.X_train,w2v.y_train
```

- Part 3:

```
#Skip gram model in python
#code part 3
corpus = ""
corpus += "The earth revolves around the sun. The moon revolves
around the earth"
epochs = 1000

training_data = preprocessing(corpus)
w2v = word2vec()

prepare_data_for_training(training_data,w2v)
w2v.train(epochs)

print(w2v.predict("around",3))
```

Output:


Skip gram model in NLP - Jupyter
localhost:8888/notebooks/Skip%20gram%20model%20in%20NLP%20.ipynb
jupyter Skip gram model in NLP Last Checkpoint: 12 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
print(w2v.predict("around",3))
epoch 185 loss = 41.108514850961775
epoch 186 loss = 41.10569064888402
epoch 187 loss = 41.10289591875256
epoch 188 loss = 41.10013020995324
epoch 189 loss = 41.09739308084281
epoch 190 loss = 41.09468409853093
epoch 191 loss = 41.09200283866863
epoch 192 loss = 41.089348885242586
epoch 193 loss = 41.086721830375126
epoch 194 loss = 41.084121274130204
epoch 195 loss = 41.08154682432431
epoch 196 loss = 41.078998096343
epoch 197 loss = 41.0764747129625
epoch 198 loss = 41.0739763041759
epoch 199 loss = 41.071502507024604
epoch 200 loss = 41.06905296543409
epoch 201 loss = 41.06662733005428
epoch 202 loss = 41.06422525810413
epoch 203 loss = 41.061846413220536
In [ ]:
Activate Windows
Go to Settings to activate Windows.
Type here to search
30°C
ENG US
9:50 PM 8/25/2021


Skip gram model in NLP - Jupyter
localhost:8888/notebooks/Skip%20gram%20model%20in%20NLP%20.ipynb
jupyter Skip gram model in NLP Last Checkpoint: 12 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
print(w2v.predict("around",3))
epoch 982 loss = 40.67877626282065
epoch 983 loss = 40.67867523085261
epoch 984 loss = 40.678574405046376
epoch 985 loss = 40.67847378477199
epoch 986 loss = 40.67837336940205
epoch 987 loss = 40.67827315831175
epoch 988 loss = 40.67817315087874
epoch 989 loss = 40.67807334648328
epoch 990 loss = 40.67797374450808
epoch 991 loss = 40.67787434433835
epoch 992 loss = 40.6777751453618
epoch 993 loss = 40.67767614696867
epoch 994 loss = 40.67757734855153
epoch 995 loss = 40.677478749505525
epoch 996 loss = 40.6773803492281 5
epoch 997 loss = 40.677282147119385
epoch 998 loss = 40.677184142581574
epoch 999 loss = 40.677086335019474
['moon', 'sun', 'revolves']
In [ ]:
Activate Windows
Go to Settings to activate Windows.
Type here to search
30°C
ENG US
9:50 PM 8/25/2021