PARTICULARS OF EXPERIMENTS PERFORMED

Serial No.	NAME OF THE EXPERIMENT	Page No.	Date of Expt.	Date of Submission	Remarks
1.	NLP models	01-03	10/11/22	17/11/22	
2.	Spell Checkers	04-05	17/11/22	24/11/22	
3.	Word frequency distribution on sentence Segmentation and word Segmentation	06 08	24/11/22	26/11/22	
4.	Count Vectorizer	09-10	26/11/22	1/12/22	
5.	N-Gram model, Bag of word, Inverse Trans-formations, TF and IDF ~~pos~~	11-15	1/12/22	8/12/22	
6.	Pos Tagging	16-17	8/12/22	15/12/22	
7.	Multiword Expressions	18	15/12/22	29/12/22	
8.	Cleaning the text data	19-20	29/12/22	4/1/23	
9.	Chunking and chinking	21-22	4/1/23	5/1/23	
10.	Matplot library	23-26	5/1/23	7/1/23	
11.	HMM Learn	27-28	7/1/23	19/1/23	
12.	Word Embedding in NLP	29	19/1/23	2/2/23	

1. NLP MODELS

Tokenization :

```
>>> import nltk
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> string = """ hi ! hello, how are you. """
>>> sent_tokenize (string)
o/p: [`hi !', `hello, how are you.']
>>> string = """hi! how are you ? ?""""
>>> sent_tokenize (string)
o/p: [`hi! how are you ?', `?']
>>> word_tokenize(string)
o/p: [`hi', `!', `how', `are', `you', `?', `?']
```

Stopwords :

```
>>> nltk.download ("stopwords")
o/p:    [nltk_data] Downloading package stopwords to
        [nltk_data] C:\users\y20cbu\AppData\Roaming\nltk_data ...
        [nltk_data] Package stopwords is already up-to-date!
        True
>>> from nltk.corpus import stopwords
>>> from nltk.tokenize import word_tokenize
>>> string = "ramu ate the apple !!"
>>> string1 = word_tokenize(string)
>>> string1
 o/p: [`ramu', `ate', `the', `apple', `!', `!']
>>> stop_words = set(stopwords.words("english"))
```

```
>>> filtered_list=[]
>>> for word in words_in_quote:
....        if word.case_fold() not in stop_words:
....        filtered_list.append(word)
>>> filtered_list=[
....word for word in strings if word.casefold()not in stop_words
....]
>>> filtered_list
```

O/P: ['ramu', 'ate', 'apple', '!', '!']

Stemming :

```
>>> from nltk.stem import PorterStemmer
>>> from nltk.tokenize import word_tokenize
>>> Stemmer = PorterStemmer()
>>> String = """helping duplicating waiting vanishing washing
    tracker"""
>>> word1 = word_tokenize(string)
>>> word1
```

O/P: ['helping', 'duplicating', 'waiting', 'vanishing', 'washing', 'tracker']

```
>>> stemmed_word1 = [stemmer.stem(word) for word in word1]
>>> stemmed_word1
```

O/P: ['help', 'duplic', 'wait', 'vanish', 'wash', 'tracker']

Tagging parts of speech:

```
>>> from nltk.tokenize import word_tokenize
>>> string = """ you wish to make an apple pie from scratch"""
>>> words = word_tokenize(string)
>>> import nltk
```

```
>>> nltk.pos_tag(words)
o/p: [('you', 'PRP'),
      ('wish', 'VBP'),
      ('to', 'TO'),
      ('make', 'VB'),
      ('an', 'DT'),
      ('apple', 'NN'),
      ('pie', 'NN'),
      ('from', 'IN'),
      ('scratch', 'NN'),
      ('.', '.')]
```

Lemmatizing:

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> lemmatizer.lemmatize("scarves")
o/p: 'scarf'
>>> string = " Aksh loves apples"
>>> words = word_tokenize(string)
>>> words
o/p: ['Aksh', 'loves', 'apples']
>>> lemmatized_words = [lemmatizer.lemmatize(word) for word in
    words]
>>> lemmatized_words
o/p: ['Aksh', 'love', 'apple']
```

2. Spell Checking:

Text Blob : (Standard Spell Checker)

Cmd:

Pip install textblob

Idle:

```
>>> from textblob import TextBlob
>>> fb = TextBlob("i want fotball")
>>> fb.correct()
o/p: TextBlob("i want football")
>>> fb = TextBlob("i want to paly fotball")
>>> fb.correct()
o/p: TextBlob("i want to pale football")
>>> fb = TextBlob("Pyhton is easys to learn")
>>> fb.correct()
o/p: TextBlob("platon is days to learn")
```

Custom Spell checker: (Spello)

Cmd:

Pip install spello

Idle:

```
>>> from spello.model impat SpellCorrectionModel
>>> sp = SpellCorrectionModel(language = "en")
```

→ Create a sample.txt file and save it in a notepad and copy the path of that file.

```
>>> with open("c:/users/y20cb004/Documents/sample.txt", "r") as file:
    data = file.readlines()
```

```
data = [i.strip() for i in data]
data
```

O/P: ['spell checking in nltk.', 'nlp experience some ambiguity at each level.', 'open cmd prompt.', 'open python and type.']

```
>>> sp.train(data)
```

O/P: Spello training started...
Context model training started...
Symspell training started...
Phoneme training started...
Spello training completed successfully.....

```
>>> sp.spell_correct("spell chekcing in nltk")
```

O/P: {'original_text': 'spell chekcing in nltk', 'spell_corrected_text': 'spell checking in nltk', 'correction_dict': {'sepll': 'spell', 'chekcing': 'checking', 'nlik': 'nltk'}}

3. Word Segmentation :

cmd : Pip install spacy

Idle :

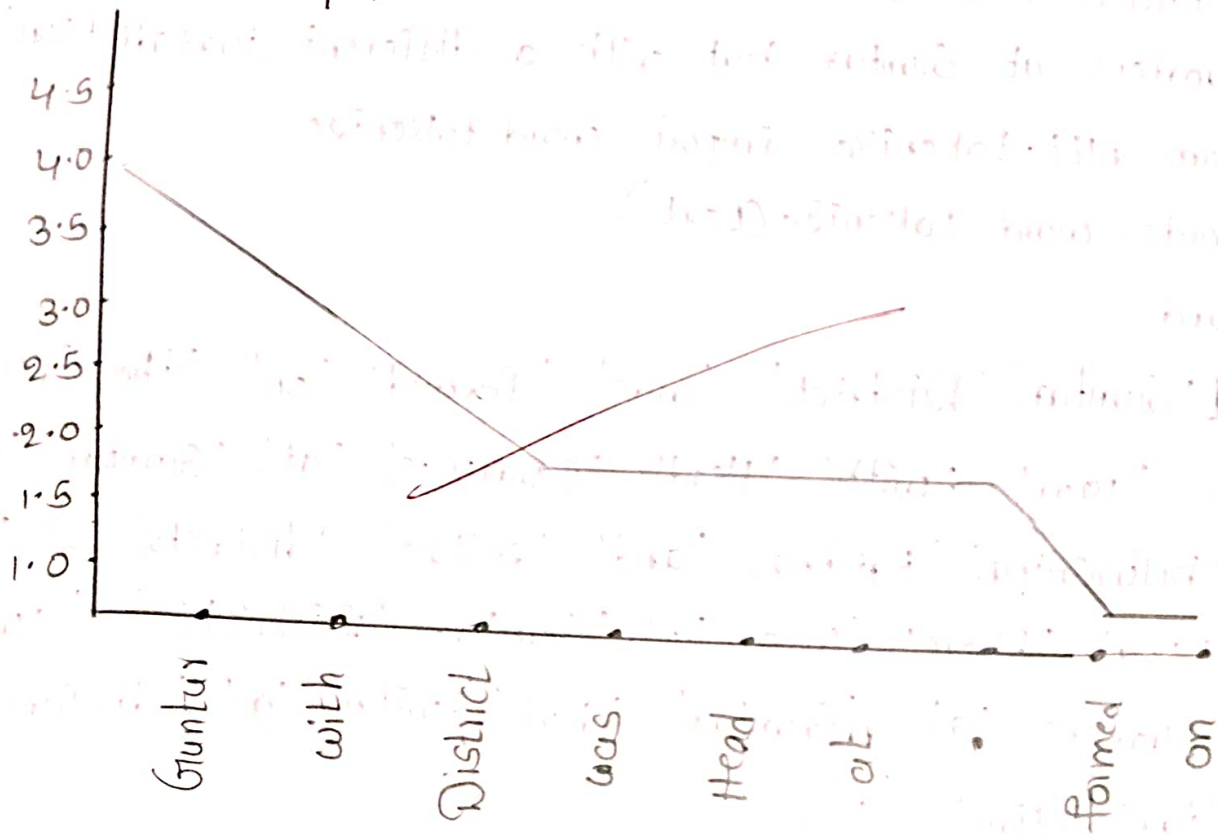
```
>>> import nltk
>>> from nltk.probability import FreqDist
>>> text = " Guntur District was formed on the 1st october, 1904 with
    Head Quarters at Guntur after bifurcating Krishna and Nellore
    districts. Prior to 1859 there was 'Guntur District' with
    Head Quarters at Guntur but with a different jurisdiction".
>>> text
```

O/P : 'Guntur District was formed on the 1st october, 1904 with Head Quarters at Guntur after bifurcating Krishna and Nellore districts. Prior to 1859 there was 'Guntur District' with Head Quarters at Guntur but with a different jurisdiction".

```
>>> from nltk.tokenize import word_tokenize
>>> words = word_tokenize(text)
>>> word
```

O/P : ['Guntur', 'District', 'was', 'formed', 'on', 'the', '1st', 'october', ',', '1904', 'with', 'Head', 'Quarters', 'at', 'Guntur', 'after', 'bifurcating', 'Krishna', 'and', 'Nellore', 'districts', '.', 'Prior', 'to', '1859', 'there', 'was', '', 'Guntur', 'District', '', 'with', 'Head', 'Quarters', 'at', 'Guntur', 'but', 'with', 'a', 'different', 'juridiction', '.']

Word Segmentation Using plot:-



```
>>> len(words)
o/p: 42
```

Word Frequency Distribution:

```
>>> FreqDist(words)
o/p: FreqDist({'Guntur': 4, 'with': 3, 'District': 2, 'was': 2, 'Head': 2,
    'Quarters': 2, 'at': 2, '.': 2, 'formed': 1, 'on': 1, ....})
>>> word1 = FreqDist(words)
>>> word1
o/p: FreqDist({'Guntur': 4, 'with': 3, 'District': 2, 'was': 2, 'Head': 2,
    'Quarters': 2, 'at': 2, '.': 2, 'formed': 1, 'on': 1, ....})
>>> Print(word1)
o/p: << FreqDist with 31 samples and 42 outcomes >>
>>> len(FreqDist(words))
o/p: 31
>>> word1.most_common(5)
o/p: [('Guntur', 4), ('with', 3), ('District', 2), ('was', 2), ('Head', 2)]
>>> word1.freq("at")
o/p: 0.047619
>>> word1.freq("mlt")
o/p: 0.0
```

Word segmentation using plot:

```
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(figsize=(10,5))
>>> word1.plot(10)
```

Sentence Segmentation :

Cmd :

pip install spacy

Python -m spacy download en-core-web-lg

Idle :

```
>>> import spacy
>>> import nltk
>>> n = spacy.load("en-core-web-lg")
>>> n
```

O/p : <spacy.lang.en.English object at 0X0000014701BF89D0>

```
>>> doc = n("This is First line. This is second line")
>>> doc
```

O/p : This is First line. This is second line

```
>>> for sent in doc.sents:
....    print(sent.text)
```

O/p : This is First line
This is second line

4. Count Vector Class

Cmd:

```
Pip install scikit-learn scipy matplotlib numpy
                    (or)
Pip install sklearn
```

Idle:

```
>>> import sklearn
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> word1 = ["This is first line, this is second line, this is third
    line"]
>>> word1
O/P: ['This is first line, this is second line, This is third line']
>>> vectorizer = CountVectorizer()
>>> vectorizer.fit(word1)
O/P: CountVectorizer()
>>> X = vectorizer.fit_transform(word1)
>>> X
O/P: <1x6 sparse matrix of type '<class 'numpy.int64'>' with 6 stored
    elements in Compressed Sparse Row format>
>>> vectorizer.vocabulary_
O/P: {'this': 5, 'is': 1, 'first': 0, 'line': 2, 'second': 3, 'third': 4}
>>> vectorizer.vocabulary_.get("this")
O/P: 5
>>> vectorizer.get_feature_names()
O/P: ['first', 'is', 'line', 'second', 'third', 'this']
```

```
>>> X.toarray()
O/P: array([[1, 3, 3, 1, 1, 3]], dtype=int64)
```

cmd:

```
Pip install pandas
```

Idle:

```
>>> import pandas as pd
>>> matrix = pd.DataFrame(X.toarray(), columns = Vectorizer.get_feature_
    names())
>>> matrix
```

O/P:

	first	is	line	second	third	this
0	1	3	3	1	1	3

5. N-Gram Model :

Idle :

```
>>> import sklearn
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> text = ["this is my new sentence"]
>>> n-gram = CountVectorizer(ngram_range=(2,2))
>>> trans = n-gram.fit-transform(text)
>>> trans
```

o/p : <1x4 sparse matrix of type '<class 'numpy.int64'>' with 4 stored elements in Compressed Sparse Row format>

```
>>> n-gram.vocabulary-
```

o/p : {'this is' : 3, 'is my' : 0, 'my new' : 1, 'new sentence' : 2}

```
>>> trans.toarray()
```

o/p : array([[1, 1, 1, 1]], dtype=int64)

```
>>> n-gram = CountVectorizer(ngram_range=(3,3))
>>> trans = n-gram.fit-transform(text)
>>> trans
```

o/p : <1x3 sparse matrix of type '<class 'numpy.int64'>' with 3 stored elements in Compressed Sparse Row format>

```
>>> n-gram.vocabulary-
```

o/p : {'this is my' : 2, 'is my new' : 0, 'my new sentence' : 1}

```
>>> trans.toarray()
```

o/p : array([[1, 1, 1]], dtype=int64)

```
>>> n-gram = CountVectorizer(ngram_range=(1,3))
>>> trans = n-gram.fit-transform(text)
```

>>> trans

o/p: <1x12 Sparse matrix of type '<class 'numpy.int64'>' with 12 stored elements in Compressed Sparse Row format>

>>> n-gram.vocabulary_

o/p: {'this': 9, 'is': 0, 'my': 3, 'new': 6, 'sentence': 8, 'this is': 10, 'is my': 1, 'my new': 4, 'new sentence': 7, 'this is my': 11, 'is my new': 2, 'my new sentence': 5}

>>> trans.toarray()

o/p: array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]], dtype=int64)

>>> n-gram.vocabulary_.get("this is")

o/p: 10

Bag of Words :

>>> trans.shape

o/p: (1, 12)

>>> t = ("i am learning nlp")

>>> t

o/p: ['i am learning nlp']

>>> n_gram.transform(t).toarray()

o/p: array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)

>>> t = ["i am my learning nlp"]

>>> n-gram.transform(t).toarray()

o/p: array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int 64)

>>> trans = n-gram.fit-transform(text+t)

>>> trans

o/p: <2x20 sparse matrix of type '<class 'numpy.int 64'>' with
21 stored elements in Compressed Sparse Row format>

>>> n-gram.vocabulary-

o/p: {'this':17, 'is':3, 'my': 8, 'new': 13, 'sentence':16, 'this is':18,
'is my':4, 'my new':11, 'new sentence':14, 'this is my':19, 'is my
new':5, 'my new sentence':12, 'am':0, 'learning':6, 'nlp':15,
'am my':1, 'my learning':9, 'learning nlp':7, 'am my learning': 2,
'my learning nlp':10}

>>> n-gram.transform(text+t).toarray()

o/p: array([[0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]], dtype = int 64)

Inverse Transformations :

Idle:

```
>>> import sklearn
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> word1 = ["Welcome to CSBS, Welcome to third year, Welcome to Natural Language Processing"]
>>> word1
```

O/P : ['Welcome to CSBS, Welcome to third year, Welcome to Natural Language Processing']

```
>>> vectorizer = CountVectorizer()
>>> vectorizer.fit(word1)
```

O/P : CountVectorizer()

```
>>> X = vectorizer.fit_transform(word1)
>>> X
```

O/P : <1x8 sparse matrix of type '<class 'numpy.int64'>' with 8 stored elements in Compressed Sparse Row format>

```
>>> vectorizer.vocabulary_
```

O/P : {'welcome': 6, 'to': 5, 'csbs': 0, 'third': 4, 'year': 7, 'natural': 2, 'language': 1, 'processing': 3}

```
>>> vectorizer.inverse_transform(X)
```

O/P : [array(['Welcome', 'to', 'csbs', 'third', 'year', 'natural', 'language', 'processing'], dtype='<U10')]

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> import pandas as pd
>>> Corpus = ["This is first sentence," "This is second sentence", "This is third sentence"]
>>> Corpus
```

O/P: ['This is first sentence', 'This is second sentence', 'This is third sentence']

>>> corpus[0]

O/P: 'This is first sentence'

>>> corpus[1]

O/P: 'This is second sentence'

>>> vectorizer = TfidfVectorizer()

>>> X = vectorizer.fit_transform(corpus)

>>> X

O/P: <3x6 sparse matrix of type '<class 'numpy.float64'>' with 12 stored elements in Compressed Sparse Row format>

>>> vectorizer.vocabulary_

O/P: {'this': 5, 'is': 1, 'first': 0, 'sentence': 3, 'second': 2, 'third': 4}

>>> vectorizer.get_feature_names()

O/P: ['first', 'is', 'second', 'sentence', 'third', 'this']

>>> vectorizer.idf_

O/P: array([1.69314718, 1. , 1.69314718, 1. , 1.69314718, 1.])

>>> matrix = pd.DataFrame(x.toarray(), columns = vectorizer.get_feature_names())

>>> matrix

O/P:

	first	is	second	sentence	third	this
0	0.69903	0.412859	0.00000	0.412859	0.00000	0.412859
1	0.00000	0.412859	0.69903	0.412859	0.00000	0.412859
2	0.00000	0.412859	0.00000	0.412859	0.69903	0.412859

6. Pos Tagging :

Idle :

```
>>> import nltk
>>> nltk.help.upenn_tagset("MD")
```

O/P : MD: modal auxiliary

can cannot could couldn't dare may might must need ought shall should shouldn't will would.

```
>>> text="The Python programming language provides a wide range of tools and libraries for attacking specific NLP tasks"
>>> text
```

O/P : 'The python programming language provides a wide range of tools and libraries for attacking specific NLP tasks'

```
>>> words = nltk.word_tokenize(text)
>>> words
```

O/P : ['The', 'python', 'programming', 'language', 'provides', 'a', 'wide', 'range', 'of', 'tools', 'and', 'libraries', 'for', 'attacking', 'specific', 'NLP', 'tasks', '.']

```
>>> len(words)
```

O/P : 18

```
>>> nltk.pos_tag(words)
```

O/P : [('The', 'DT'), ('Python', 'NNP'), ('programming', 'NN'), ('language', 'NN'), ('provides', 'VBZ'), ('a', 'DT'), ('wide', 'JJ'), ('range', 'NN'), ('of', 'IN'), ('tools', 'NNS'), ('and', 'CC'), ('libraries', 'NNS'), ('for', 'IN'), ('attacking', 'VBG'), ('specific', 'JJ'), ('NLP', 'NNP'), ('tasks', 'NNS'), ('.', '.')]

```
>>> nltk.help.brown_tagset()
```

o/p:

(: opening parenthesis
(

): closing parenthesis
)

*: negator
not n't

, : Comma

--: dash
--

. : sentence terminator

: : colon ? ; ! :

ABN: determiner / pronoun, pre-qualifier
all half many nary

WRB + MD: WH-adverb + modal auxillary
where'd

>>> nltk.help.upenn_tagset("NNP")

o/p: NNP: noun, proper, singular
Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christas Occanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Helter Liverpool.....

>>> nltk.help.upenn_tagset("cc")

o/p: cc: Conjunction, Coordinating
&'n and both but either etc for less minus neither nor or plus so Therefore Times v. versus vs. whether yet

etc

Multiword Expressions

```
>>> import nltk
>>> nltk.download('punkt')
o/p: [nltk_data] Downloading package punkt to
     [nltk_data] C:\Users\420cb4\AppData\Roaming\
      nltk_data....
     [nltk_data] Package punkt is already up-to-date!
     True
>>> from nltk.tokenize import MWETokenizer
>>> from nltk.tokenize import word_tokenize
>>> mwe = MWETokenizer([('new', 'york'), ('Hang', 'Kong'), ('takes', 'off')],
    separator = "_")
>>> text1 = "the flight is going from new york to Hang kong"
>>> text2 = "the plane takes off from new york at 10am"
>>> mwe1 = mwe.tokenize(word_tokenize(text1))
>>> mwe2 = mwe.tokenize(word_tokenize(text2))
>>> mwe1
o/p: ['the', 'flight', 'is', 'going', 'from', 'new', 'york', 'to', 'Hang',
      'Kong']
>>> mwe2
o/p: ['the', 'plane', 'takes-off', 'from', 'new', 'york', 'at', '10am']
```

8 Cleaning Text Data

cmd:

Pip install cleantext

Idle:

```
>>> impoat cleanLext
>>> from cleantext impoat clean
```

Ascii Code:

```
>>> s1 = "Ko\u017eu\u0161\u010idek"
>>> clean(s1, Lo_ascii='True')
```

o/p: 'Kozusadck'

LowerCase:

```
>>> s2 = " my name is Chandu"
>>> clean(s2, lower = 'True')
```

o/p: 'my name is chandu'

Url Replacement:

```
>>> s3 = " hLLps://www.google.com has surpassed hLtps://www.Bing.Com in
    search volume"
>>> clean(s3, no-urls = 'True', replace_with_url = "URL")
```

o/p: 'Url has surpassed url in search volume'

Replace Currency:

```
>>> s4 = " i want $40"
>>> clean(s4, no-Currency-symbols = 'True')
```

o/p: 'i want <cur> 40'

```
>>> clean(s4, no-Currency-symbols = 'True', replace-with-Currency-Symbol
    = "Rupees")
```

o/p: 'i want rupees40'

No Punctuations:

```
>>> s5 = "40,000 is more than 30,000"
>>> clean(s5, no-punct = 'True')
o/p: '40000 is more than 30000'
>>> clean(s5, no-punct = 'True', replace-with-punct = "@")
o/p: '40@000 is more than 30@000'
```

Digit Replacement:

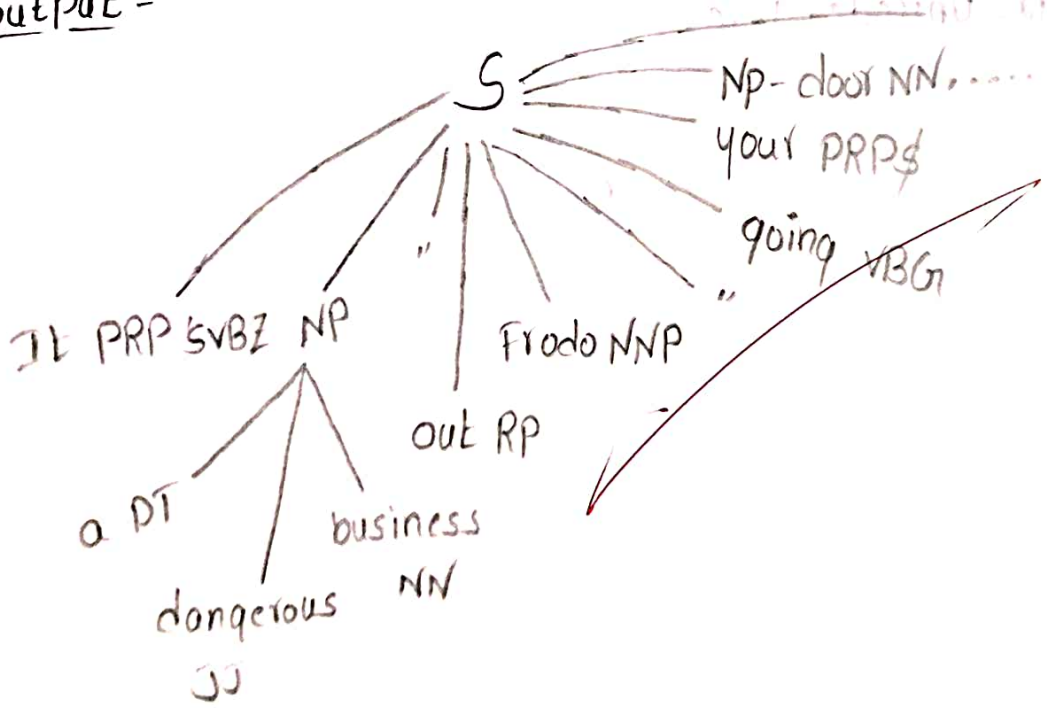
```
>>> s6 = "hleghty34345676jyhtuiop"
>>> clean(s6, no-digits = 'True')
o/p: 'hleghtyoooooooojyhtuiop'
>>> clean(s6, no-digits = 'True', replace_with_digit = " ")
o/p: 'hleghty jyhtuiop'
```

Unicode:

```
>>> s7 = "zürick"
>>> clean(s7, fix_unicode = True)
o/p: 'zurick'
```

9. Chunking :

output -



9. Chunking :

Idle :

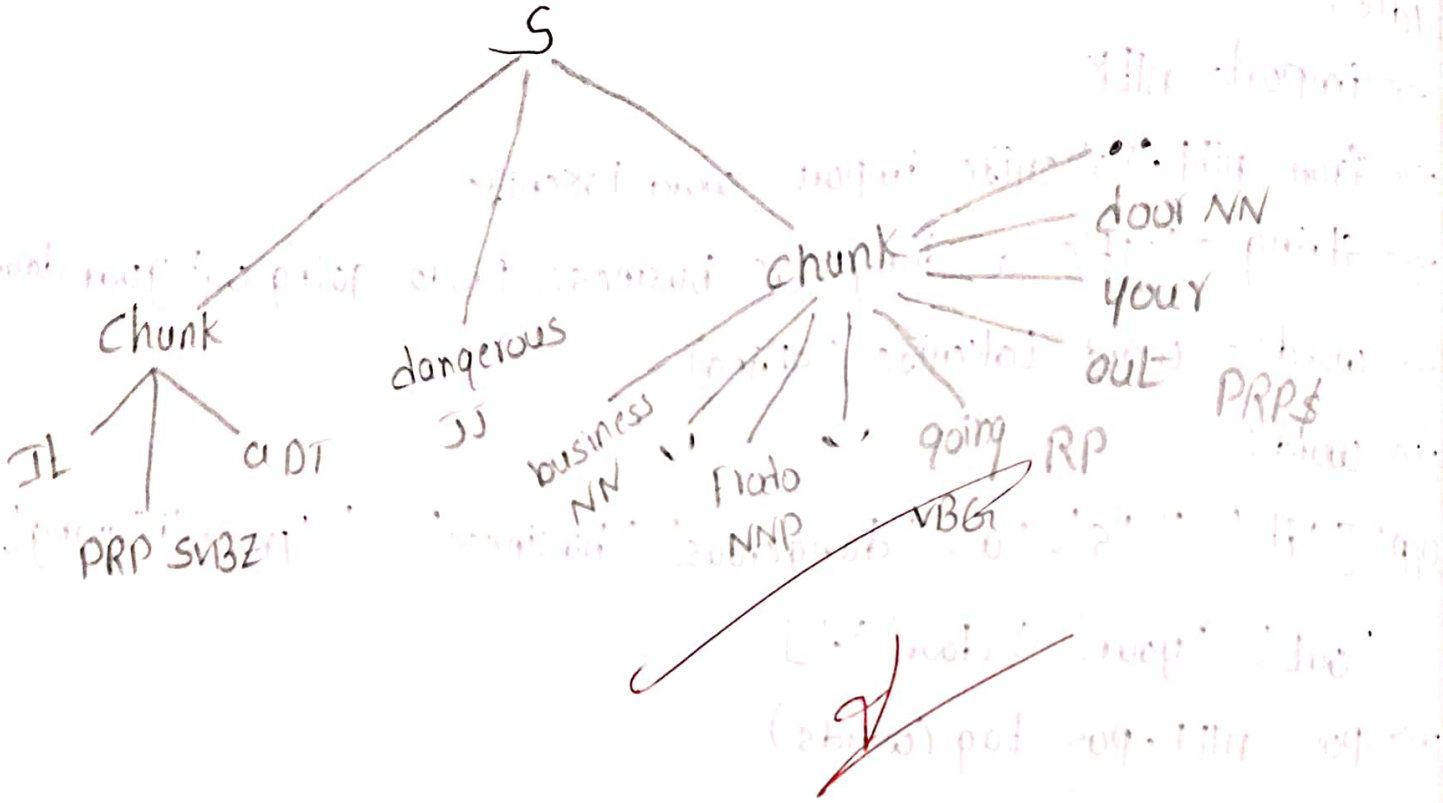
```
>>> import nltk
>>> from nltk.tokenize import word_tokenize
>>> String = "It's a dangerous business, Frodo, going out your door."
>>> words = word_tokenize (string)
>>> words
o/p: ['it', "'s", 'a', 'dangerous', 'business', ',', 'Frodo', ',', 'going',
     'out', 'your', 'door', '.']
>>> pos = nltk.pos_tag (words)
>>> pos
o/p: [('it', 'PRP'), ("'s", 'VBZ'), ('a', 'DT'), ('dangerous', 'JJ'),
     ('business', 'NN'), (',', ','), ('Frodo', 'NNP'), (',', ','),
     ('going', 'VBG'), ('out', 'RP'), ('your', 'PRP$'), ('door', 'NN'),
     ('.', '.')]
>>> grammer = "NP: {<DT>? <JJ>* <NN>}"
>>> chunk_paeser = nltk.RegexpParser(grammer)
>>> tree = chunk_paeser.parse(pass)
>>> tree.draw()
```

Chinking :

```
>>> pos
>>> grammer = " " "
```

Chinking :

output -



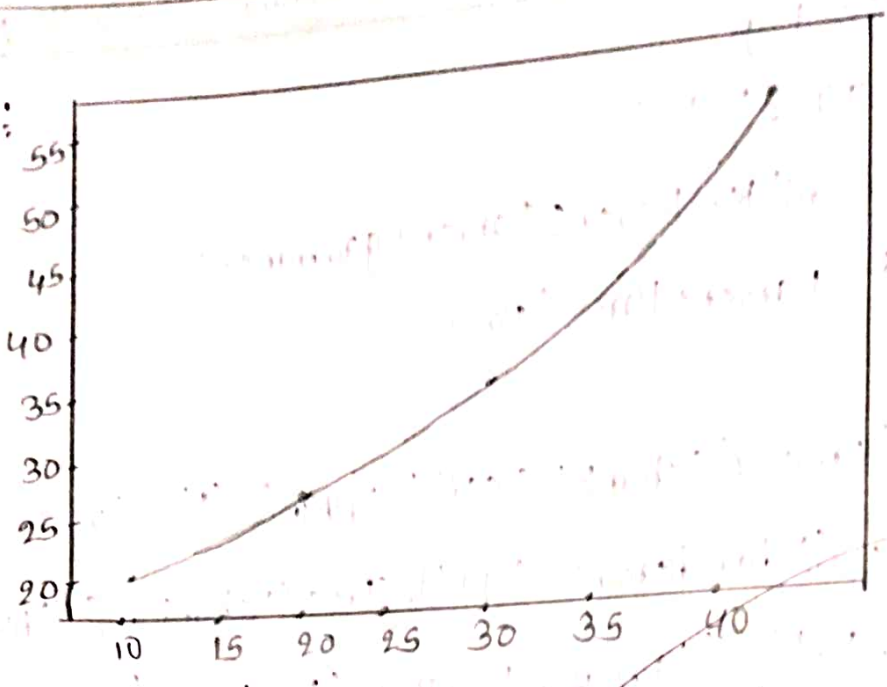
```
chunk : {<.*>+}
        }<JJ{" " "
>>> chunk-parser = nltk.RegexpParser(grammer)
>>> tree = chunk-parser.parse(pos)
>>> tree
```

O/P: Tree('S', [Tree('chunk', [('it', 'PRP'), ('''s', 'VBZ'), ('a', 'DT')]), ('dangerous', 'JJ'), Tree('chunk', [('business', 'NN'), (',', ','), ('Frodo', 'NNP'), (',', ','), ('going', 'VBG'), ('out', 'RP'), ('your', 'PRP$'), ('door', 'NN'), ('.', '.')])])

```
>>> tree.draw()
```

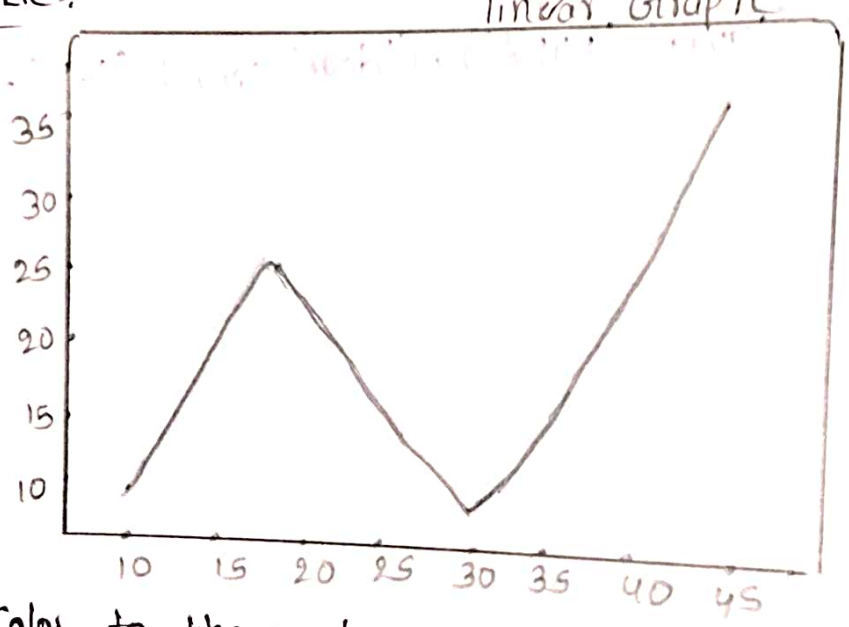
10

output:



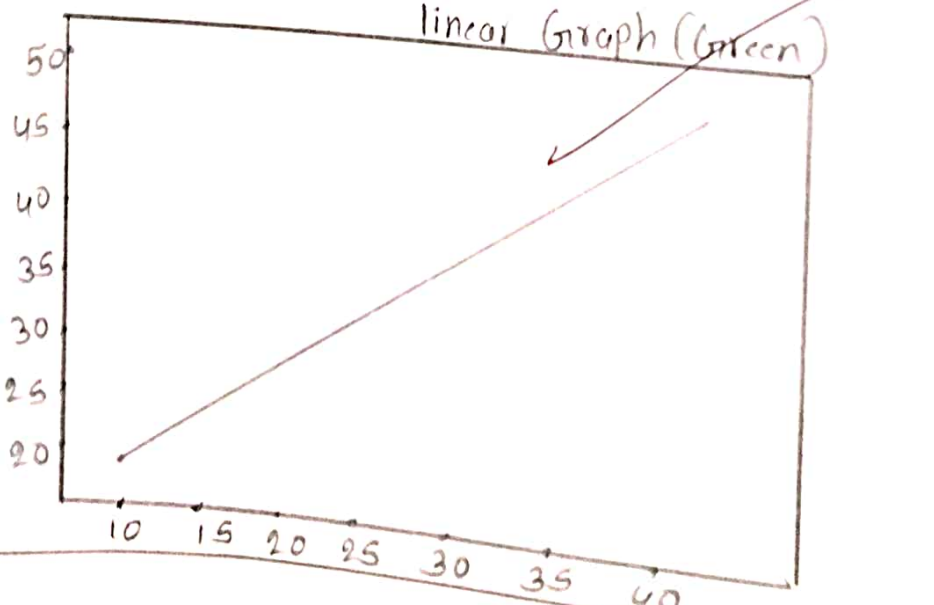
Adding title:

output:



Adding Color to the graph

output:



10. Matplot Library :

Cmd :

```
Pip install matplotlib
```

Idle :

```
>>> import matplotlib.pyplot as plt
>>> x = [10,20,30,40]
>>> y = [20,25,35,55]
>>> plt.plot(x,y)
o/p: [<matplotlib.lines.Line2D object at 0x00016>]
>>> plt.show()
```

Adding title :

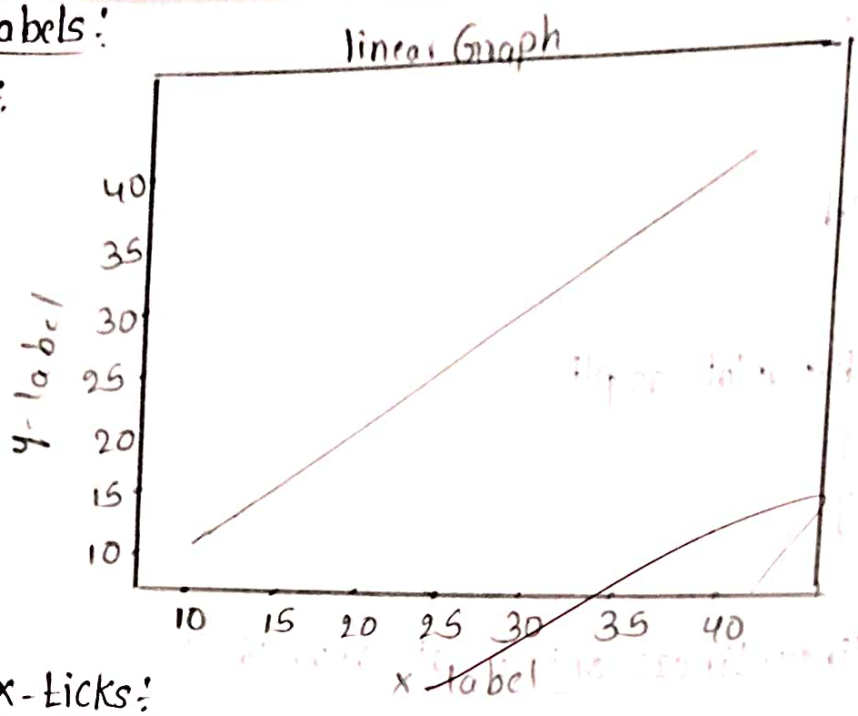
```
>>> x = [10,20,30,40]
>>> y = [20,25,20,35]
>>> plt.plot(x,y)
o/p: [<matplotlib.lines.Line2D object at 0x00>]
>>> plt.title("linear Graph")
o/p: Text(0.5,1.0,'linear Graph')
>>> plt.show()
```

Adding color to the graph :

```
>>> x = [10,20,30,40]
>>> y = [20,30,40,50]
>>> plt.plot(x,y)
o/p: [<matplotlib.lines.Line2D object at 0x000>]
>>> plt.title("linear Graph",fontsize=25,color="green")
o/p: Text(0.5,1.0,'linear Graph')
>>> plt.show()
```

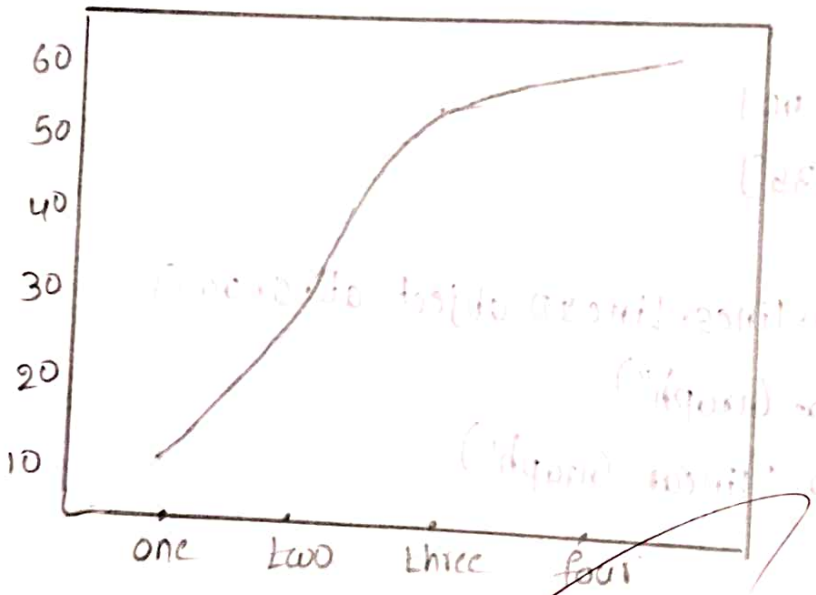
Adding labels:

output:



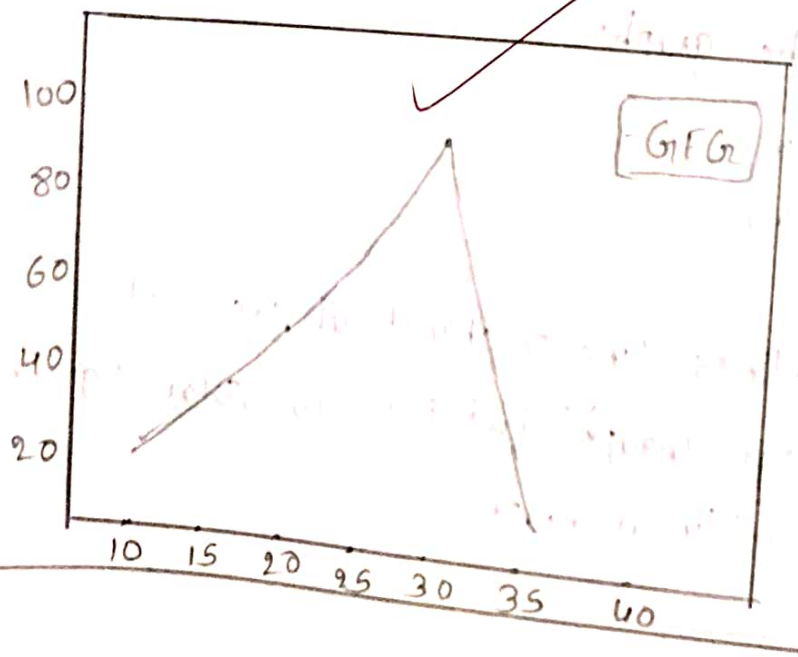
ylim and x-ticks:

output:



legend:

output:



Adding labels :

```
>>> x = [10,20,30,40]
>>> y = [20,30,40,50]
>>> plt.plot(x,y)
>>> plt.title('linear Graph', fontsize=25, color="green")
>>> plt.xlabel('x-label')
>>> plt.xlabel('y-label')
>>> plt.show()
```

ylim and xticks :

```
>>> x = [10,20,30,40]
>>> y = [20,30,40,50]
>>> plt.ylim(0,60)
>>> plt.xticks(x, labels = ["one", "two", "three", "four"])
>>> plt.plot(x,y)
>>> plt.show()
```

legend :

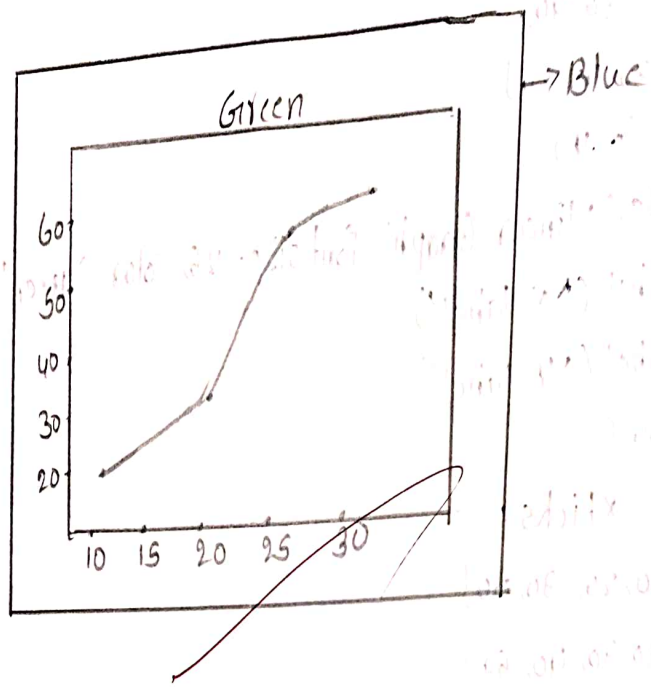
```
>>> x = [10,20,30,40]
>>> y = [20,34,65,34]
>>> plt.plot(x,y)
>>> plt.title('Linear Graph')
>>> plt.xlabel('x label')
>>> plt.ylabel('y-label')
>>> plt.legend(["GFG"])
>>> plt.show()
```

Figure class :

```
>>> import matplotlib.pyplot as plt
>>> from matplotlib.figure import Figure
>>> x = [10,20,30,40]
>>> y = [20,25,35,55]
# Creating a new figure with width = 7 inches and height = 5 inches
```

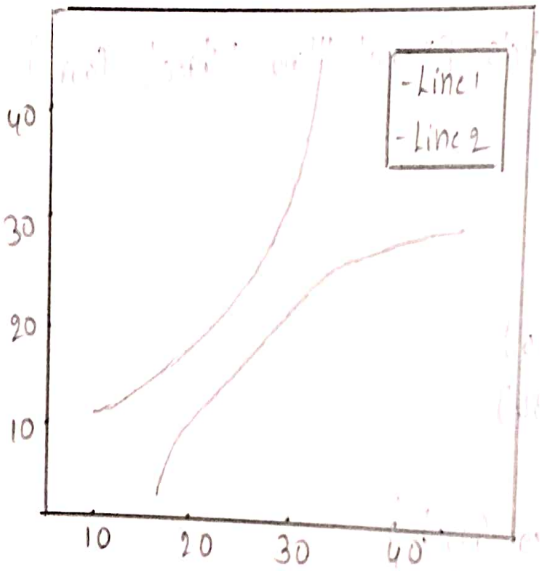
Figure class:

output:



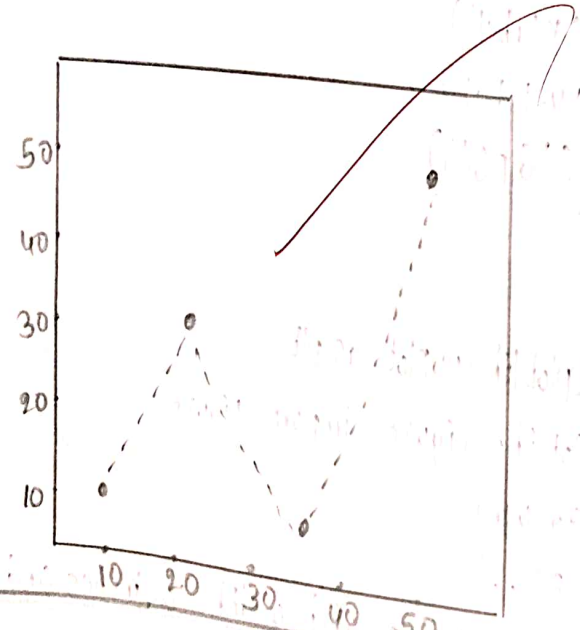
Multiple plots:

output:



Linechart:

output:



with face color as Green, edgecolor as red and line with edge as 7

```
>>> fig = plt.figure (figsize = (7,5), face color = 'g', edgecolor = 'b';
    linewidth = 7)
# Creating new axes for the figure
>>> ax = fig.add_axes ([1,1,1,1])
# Adding the data to be plotted
>>> ax.plot (x,y)
```

Multiple plots :

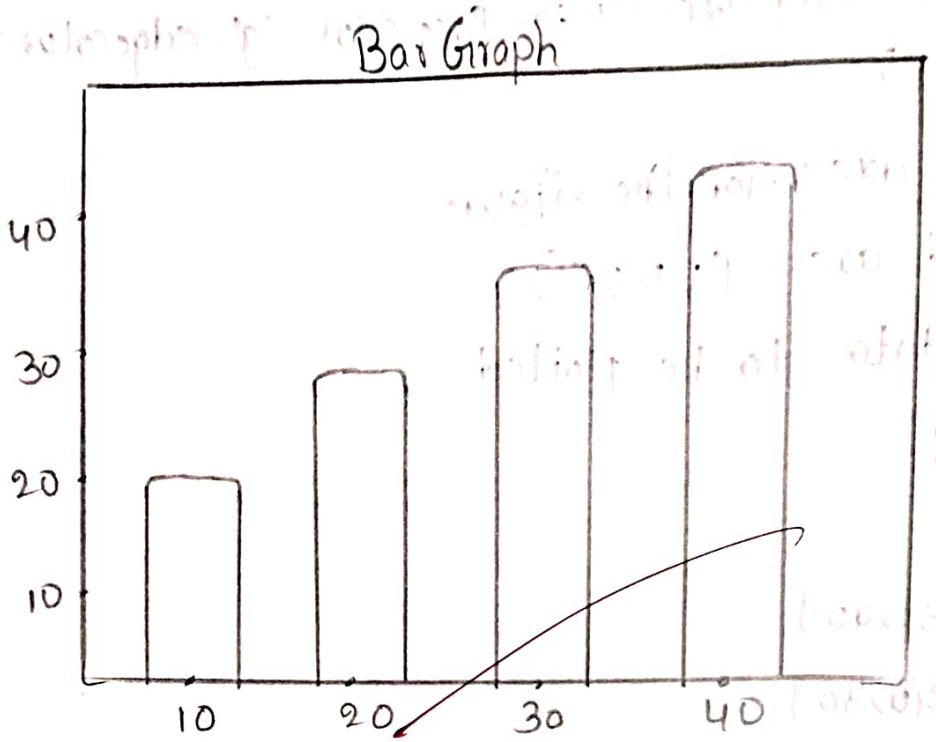
```
>>> x = [10, 20, 30, 40]
>>> y = [10, 20, 40, 30]
>>> fig = plt.figure (figsize = (5,4))
>>> ax = fig.add_axes ([1,1,1,1])
>>> ax1 = ax.plot (x,y)
>>> ax2 = ax.plot (y,x)
>>> ax.set_title ("Multiple Graphs")
>>> ax.set_xlabel ("x-Axis")
>>> ax.set_ylabel ("y-Axis")
>>> ax.Legend (labels = ['Line 1', "Line 2"])
>>> plt.show()
```

Line chart :

```
>>import matplotlib.pyplot as plt
>>> x = [10, 20, 30, 40]
>>> y = [20, 25, 30, 35]
>>> plt.plot (x, y, color = 'green', linewidth = 3, marker = 'o', markersize = 15,
    linestyle = '--')
>>> plt.title ("Line chart")
>>> plt.xlabel ("x-axis")
```

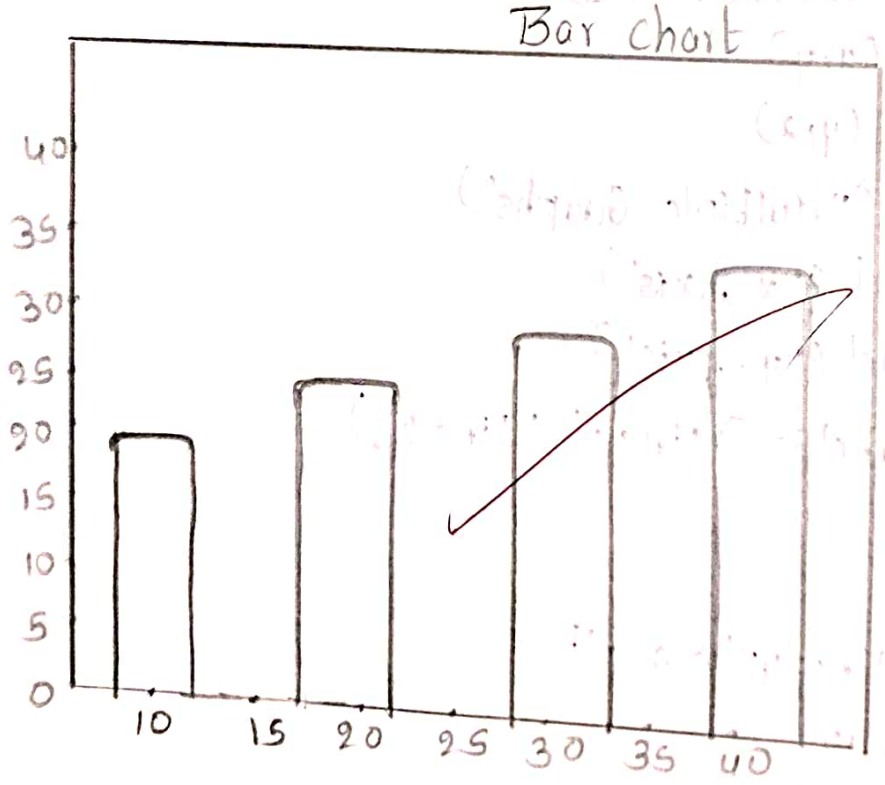
Bar Graph:

output:



Bar chart:

output:



plt.ylabel ("y-axis")
>>> plt.show()

Bar Graph:

```
>>> import matplotlib.pyplot as plt
>>> x = [10, 20, 30, 40]
>>> y = [20, 30, 40, 50]
>>> plt.bar(x, y, color = 'green', width = 3)
>>> plt.title ("Bar Graph")
>>> plt.show()
```

Bar Chart:

```
>>> import matplotlib.pyplot as plt
>>> x = [10, 20, 30, 40]
>>> y = [20, 30, 40, 50]
>>> plt.bar(x, y, color = "green", width = 3, edgecolor = 'blue', linewidth = 2)
>>> plt.title ("Barchart")
>>> plt.show()
```

HMM Learn :

Cmd :

pip install hmmlearn

Idle :

```
>>> import nltk
>>> import hmmlearn
>>> import numpy as np
>>> from hmmlearn.hmm import CategoricalHMM
>>> startprob = np.array([0.5, 0.5])
>>> transmat = np.array([[0.7, 0.3], [0.3, 0.7]])
>>> covar = np.array([[0.9, 0.1], [0.2, 0.8]])
>>> model = CategoricalHMM(n_components=2, startprob_prior=startprob,
    transmat_prior=transmat)
>>> x = [[0, 1, 0, 1], [0, 0, 0, 0], [1, 1, 1, 1], [1, 0, 0, 0]]
>>> Model.fit(x)
```

o/p: CategoricalHMM(n_components=2, random_state=RandomState
(MT19937) at 0x19EA1586640,
startprob_prior = array([0.5, 0.5])
transmat_prior = array([[0.7, 0.3], [0.3, 0.7]])

```
>>> print(model.transmat_)
```

o/p: [[0.2800 0.7199]
[0.594 0.4053]]

```
>>> prob = model.decode(np.array([0, 1, 0, 1]).reshape(4, 1), lengths
    = None, algorithm = None)
```

```
>>> print(prob)
o/p: (-2.941, array([[1,0,1,0], dtype=int64)]
>>> print(np.exp(prob[0]))
o/p: 0.0527
>>> x,z = model.sample(10)
>>> print(x)
o/p: [[0]
      [0]
      [1]
      [0]
      [1]
      [0]
      [0]
      [1]
      [0]
      [0])
>>> print(z)
o/p: [1 1 0 1 0 0 1 0 1 1]
```

12. Word Embedding in NLP (Word2vec)

Cmd:

pip install gensim

Idle:

```
>>> import gensim
>>> from gensim.models import Word2vec
>>> import warnings
>>> warnings.filterwarnings('ignore')
>>> st=[['this','is','sample1'],['this','is','sample2'],['this','is',
        'sample3']]
>>> print(st)
```

o/p: `[['this', 'is', 'sample1'],`
`['this', 'is', 'sample2'],`
`['this', 'is', 'sample3']]`

```
>>> model = Word2vec(st, min_count=1)
>>> print(model)
```

o/p: Word2vec(vocab=5, vector_size=100, alpha=0.025)

```
>>> words = list(model.wv.key_to_index)
>>> print(words)
```

o/p: `['is', 'this', 'sample3', 'sample2', 'sample1']`