LAB 1: Prolog Basics

Title: Prolog Basics

Prolog is a logic-based programming language with primary applications in artificial intelligence and computational linguistics. "Prolog" is an abbreviation for "Programming in Logic." Prolog takes a declarative approach to problem solving, in which you express the desired outcome or objective, and the language's inference engine figures out how to attain that goal using a collection of facts and rules.

Consider the following basic example to see how Prolog works:

Code:

```
% Facts
likes( rajani, mango).
likes( rajani, ice_cream).
likes( suhana, banana).
likes( suhana, ice_cream).
likes( samir, salad).
likes( samir, chocolate).
% Rules
friend(X, Y) :- likes(X, Z), likes(Y, Z).
```

Output:

```
% c:/users/noice/onedrive - tribhuvan university/documents/prolog/rajani
 -2 clauses
?- likes(rajani, mango).
true.

?- likes(suhana, salad).
false.

?- likes(suhana, ice_cream).
true.

?- likes(samir, banana).
false.

?- friends(suhana, rajani).
Correct to: "friend(suhana,rajani)"?
Please answer 'y' or 'n'? yes
true .
```

LAB 2: Ancestor Problem

Title: Ancestor Problem

The ancestor problem is a classic example used in Prolog to demonstrate how logical inferenceandrecursive rules can be employed to solve a genealogical relationship problem. Given a set of factsabout parent-child relationships, the goal is to determine if one individual is an ancestor of another.

Here's an example of how you can define the facts and rules in Prolog to solve the ancestor problem:

Code:

```
% Facts
parent(hary, rajani).
parent(rajani, rohit).
parent(rajani, samir ).
parent(rekha, samir).
parent(rohit, rekha).
% Rules
```

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

Output:

```
Warning:    Singleton variables: [Z,Y]
% c:/users/noice/onedrive - tribhuvan university/documents/prolog/rajani
 1 clauses
?- ancestor(X, rajani).
X = hary ,

?- ancestor(rekha, samir).
true.
```

Lab 3 : Our own ancestor problem

Title: Our own ancestor problem

Code:

```
% Facts
male(rohan).
male(tom).
female(kiara).
female(rajani).
female(roji).
female(dilkumari).
male(yubaraj).
parent(tom, rohan).
parent(kiara, rajani).
parent(tom, roji).
parent(kiara, roji).
parent(dilkumari, tom).
parent(yubaraj, tom).
% Rules
father(F,C):-parent(F,C),male(F).
mother(M,C):-parent(M,C), female(M).
grandparent(GP, GC):-parent(GP,X),parent(X,GC).
ancestor(A,D):-parent(A,D).
```

Output:

```
% c:/users/noice/onedrive - tribhuvan university/documents/prolog/rajani
?-
|    mother(M, rajani).
M = kiara.

?- grandparent(GP, rohit).
false.

?- father(F, roji).
F = tom ,

?- parent(P,tom).
P = dilkumari
```

LAB 4: TOH problem using prolog

Title : TOH problem using prolog

Code :

```
% Solve Tower of Hanoi problem
solve_toh(NumDisks) :-
move(NumDisks, left, center, right).
% Base case: Move 1 disk from Source to Destination
move(1, Source, _, Destination) :-
write('Move top disk from '),
write(Source),
write(' to '),
write(Destination),
nl.
% Recursive case: Move N disks from Source to Destination using Auxiliary peg
move(N, Source, Auxiliary, Destination) :-
N > 1,
M is N - 1,
move(M, Source, Destination, Auxiliary),
move(1, Source, _, Destination),
move(M, Auxiliary, Source, Destination).
```

Output:

```
% c:/users/noice/onedrive - tribhuvan university/documents/prolog/ai compiled
lauses
?- solve_toh(2).
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
true
```

LAB 5: Factorial Problem using prolog

Title : Factorial Problem using prolog

Factorial calculations are often encountered in various mathematical and computational contexts, such as combinatory, probability theory, and algorithm analysis. **Code :**

```
% Base case: Factorial of 0 is 1
factorial(0, 1).
% Recursive case: Factorial of N is N multiplied by factorial of N-1
factorial(N, Result) :-
N > 0,
N1 is N - 1,
factorial(N1, Result1),
Result is N * Result1.
```

Output:

```
?-
% c:/users/noice/onedrive - tribhuvan university/documents/prolog/rajani
 0 clauses
?- factorial(7, Result).
Result = 5040 ,

?- factorial(8, Result).
Result = 40320 ,

?- factorial(0, Result).
Result = 1
```

LAB 6: BFS Search implementation

Title : BFS Search implementation

Source Code:

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)
    while queue:
        vertex = queue.popleft()
        print(vertex, end=" ")
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
print("BFS traversal starting from vertex 'A':")
bfs(graph, 'A')
```

Output:

```
[Running] python -u "c:\Users\Noice\OneDrive - Trib
University\Desktop\Rajani AI\bfs.py"
BFS traversal starting from vertex 'A':
A B C D E F
```

LAB 7: DFS Search implementation

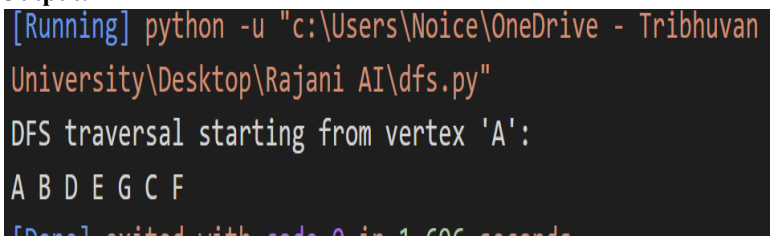
Title : DFS Search implementation

Source Code:

```
def dfs(graph, start, visited=None):
```

```
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")  # Process the current vertex (e.g., print it)

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': ['G'],
    'G': []
}
print("DFS traversal starting from vertex 'A':")
dfs(graph, 'A')
```

Output:



LAB 8: A* Search implementation

Title : A* Search implementation

Source code:

```
from queue import PriorityQueue
def heuristic(node, goal):
    distances = {
        'A': 7,
        'B': 6,
        'C': 5,
        'D': 4,
        'E': 3,
        'F': 2,
        'G': 0
    }
    return distances[node]
def astar_search(graph, start, goal):
    visited = set()
    queue = PriorityQueue()
    queue.put((0, start, [start]))
     while not queue.empty():
        cost, current_node, path = queue.get()
        if current_node == goal:
            return path
      if current_node not in visited:
```

```
            visited.add(current_node)
            for neighbor in graph[current_node]:
                new_cost = cost + graph[current_node][neighbor]
                priority = new_cost + heuristic(neighbor, goal)
                queue.put((priority, neighbor, path + [neighbor]))
    return None
graph = {
    'A': {'B': 2, 'C': 3},
    'B': {'A': 2, 'D': 4, 'E': 3},
    'C': {'A': 3, 'F': 2},
    'D': {'B': 4},
    'E': {'B': 3, 'F': 1},
    'F': {'C': 2, 'E': 1, 'G': 5},
    'G': {'F': 5}
}
start = 'A'
goal = 'G
print("A* search from vertex 'A' to 'G':")
path = astar_search(graph, start, goal)
if path is not None:
    print("Path:", path)
else:
    print("No path found.")
```

Output:

```
[Running] python -u "c:\Users\Noice\OneDrive - Tribhuvan
University\Desktop\Rajani AI\asearch.py"
A* search from vertex 'A' to 'G':
Path: ['A', 'C', 'F', 'G']
```

LAB 9: GBFS Search implementation

Title: GBFS Search implementation

Source code:

```
from queue import PriorityQueue
def heuristic(node, goal):
    # Define the heuristic function (distance) between two nodes
    # Here, we can use a dictionary to specify the distances between nodes
    distances = {
        'A': 7,   'B': 6,  'C': 5,   'D': 4, 'E': 3,   'F': 2,  'G': 0
}
    return distances[node]
def gbfs_search(graph, start, goal):
    visited = set()
    queue = PriorityQueue()
    queue.put((heuristic(start, goal), start, [start]))
    while not queue.empty():
        _, current_node, path = queue.get()
        if current_node == goal:
            return path
        if current_node not in visited:
            visited.add(current_node)
            for neighbor in graph[current_node]:
```

```
            if neighbor not in visited:
                queue.put((heuristic(neighbor, goal), neighbor, path + [neighbor]))
    return None
# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E', 'G'],
    'G': ['F']
}
start = 'A'
goal = 'G'
print("GBFS search from vertex 'A' to 'G':")
path = gbfs_search(graph, start, goal)
if path is not None:
    print("Path:", path)
else:
    print("No path found.")
```

Output:

```
[Running] python -u "c:\Users\Noice\OneDrive - Tribhuvan
University\Desktop\Rajani AI\gbfs.py"
GBFS search from vertex 'A' to 'G':
Path: ['A', 'C', 'F', 'G']
```

LAB 10: Solve 8-Queen problem

Title: Solve 8-Queen problem

Source code:

```
def is_safe(board, row, col):
    # Check if it's safe to place a queen at the given position
    # Check row and column
    for i in range(len(board)):
        if board[row][i] == 1 or board[i][col] == 1:
            return False

    # Check diagonals
    for i in range(len(board)):
        for j in range(len(board)):
            if (i + j == row + col) or (i - j == row - col):
                if board[i][j] == 1:
                    return False

    return True

def solve_queen_problem(board, col):
    # Base case: If all queens are placed, return True
    if col >= len(board):
```

```
        return True

    # Try placing a queen in each row of the current column
    for i in range(len(board)):
        if is_safe(board, i, col):
            # Place the queen at (i, col)
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solve_queen_problem(board, col + 1):
                return True

            # If placing the queen leads to an invalid solution, backtrack
            board[i][col] = 0

    # If no solution is found, return False
    return False

def print_solution(board):
    # Print the board configuration
    for row in board:
        for cell in row:
            print(cell, end=" ")
        print()

# Create an 8x8 chessboard
board = [[0] * 8 for _ in range(8)]

# Solve the 8-Queen problem
if solve_queen_problem(board, 0):
    print("Solution:")
    print_solution(board)
else:
    print("No solution found.")
```

Output:

```
[Running] python -u "c:\Users\Noice\OneDrive - Tribhuvan
University\Desktop\Rajani AI\queen.py"
Solution:
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```

LAB 11: Solve cryptoarithmetic problem

Title : Solve cryptoarithmetic problem

Source code

```
def solve_cryptoarithmetic(puzzle):
    # Get unique characters from the puzzle
    chars = set()
    for word in puzzle:
```

```
            chars.update(word)

        chars = list(chars)
        letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        digits = range(10)

        # Generate all possible digit assignments
        for perm in itertools.permutations(digits, len(chars)):
            mapping = dict(zip(chars, perm))

            # Check if the assignment satisfies the puzzle
            if is_solution(puzzle, mapping):
                # Print the solution
                print_solution(puzzle, mapping)
                return True

        # No solution found
        return False

    def is_solution(puzzle, mapping):
        # Convert words to numbers based on the digit mapping
        numbers = []
        for word in puzzle:
            number = 0
            for char in word:
                number = number * 10 + mapping[char]
            numbers.append(number)

        # Check if the sum condition holds
        return sum(numbers[:-1]) == numbers[-1]

    def print_solution(puzzle, mapping):
        # Print the puzzle solution
        for word in puzzle:
            for char in word:
                print(mapping[char], end=" ")
            print()

    # Example puzzle: SEND + MORE = MONEY
    puzzle = ["SEND", "MORE", "MONEY"]

    print("Cryptoarithmetic puzzle solution:")
    if not solve_cryptoarithmetic(puzzle):
        print("No solution found.")
```

Output:

```
[Running] python -u "c:\Users\Noice\OneDrive - Tribhuvan
University\Desktop\Rajani AI\puzzle.py"
Cryptoarithmetic puzzle solution:
2 8 1 7
0 3 6 8
0 3 1 8 5
```

Lab 12: Machine Learning using python

Title 1: Example using nltk for preprocessing text.

Example 1

Source code:

```
!pip install -q wordcloud
import wordcloud
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
import pandas as pd
import matplotlib.pyplot as plt
import io,re
import unicodedata
import numpy as np
import string
```

Output:

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Example 2:

Source code:

```
# Constants
# POS (Parts Of Speech) for: nouns, adjectives, verbs and adverbs
DI_POS_TYPES = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}
POS_TYPES = list(DI_POS_TYPES.keys())

# Constraints on tokens
MIN_STR_LEN = 3
RE_CALID = ‘[a-zA-Z]‘
```

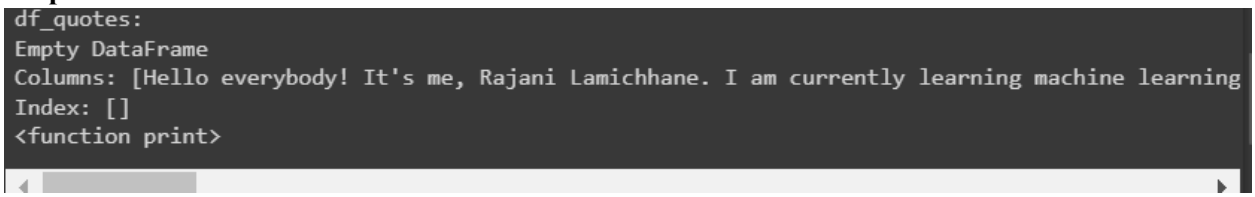
Title2: Load the corpus file from Google Drive

Source code:

```
# Upload from google drive
from google.colab import files
file_name = "Rajani AI Lab.txt"  # Update the file name or use the correct file name
df_quotes = pd.read_csv(file_name, sep='\t', encoding='utf-8')
# Display
```

```
print("df_quotes:")
print(df_quotes.to_string(index=False))  # Print the entire DataFrame without row index

# Convert quotes to list
li_quotes = df_quotes.columns.tolist()  # Convert the column names to a list
print
```

Output:



Title.3: Tokenize sentences and words, remove stopwords, use stemmer & lemmatizer First, a note on the difference between Stemming vs Lemmatization:

Stemming: Trying to shorten a word with simple regex rules
Lemmatization: Trying to find the root word with linguistics rules (with the use of regex rules

Source code:

```
# Get stopwords, stemmer and lemmatizer
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.PorterStemmer()
lemmatizer = nltk.stem.WordNetLemmatizer()

# Remove accents function
def remove_accents(data):
return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in string.ascii_letters or x == " ")

# Process all quotes
li_tokens = []
li_token_lists = []
li_lem_strings = []
for i,text in enumerate(li_quotes):

# Tokenize by sentence, then by lowercase word
tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in
nltk.word_tokenize(sent)]
# Process all tokens per quote
li_tokens_quote = []
li_tokens_quote_lem = []
for token in tokens:

# Remove accents
t = remove_accents(token)

# Remove punctuation
t = str(t).translate(string.punctuation)
li_tokens_quote.append(t)

# Add token that represents "no lemmatization match"
li_tokens_quote_lem.append("-") # this token will be removed if a lemmatization match
is found below
```

```
# Process each token
if t not in stopwords:
if re.search(RE_VALID, t):
if len(t) >= MIN_STR_LEN:

# Note that the POS (Part Of Speech) is necessary as input to the lemmatizer
# (otherwise it assumes the word is a noun)
pos = nltk.pos_tag([t])[0][1][:2]
pos2 = 'n' # set default to noun
if pos in DI_POS_TYPES:
pos2 = DI_POS_TYPES[pos]
stem = stemmer.stem(t)
lem = lemmatizer.lemmatize(t, pos=pos2) # lemmatize with the correct POS

if pos in POS_TYPES:
li_tokens.append((t, stem, lem, pos))

# Remove the "-" token and append the lemmatization match
li_tokens_quote_lem = li_tokens_quote_lem[:-1]
li_tokens_quote_lem.append(lem)

# Build list of token lists from lemmatized tokens
li_token_lists.append(li_tokens_quote)

# Build list of strings from lemmatized tokens
str_li_tokens_quote_lem = ' '.join(li_tokens_quote_lem)
li_lem_strings.append(str_li_tokens_quote_lem)

# Build resulting dataframes from lists
df_token_lists = pd.DataFrame(li_token_lists)
print("df_token_lists.head(5):")
print(df_token_lists.head(5).to_string())

# Replace None with empty string
for c in df_token_lists:
if str(df_token_lists[c].dtype) in ('object', 'string_', 'unicode_'):
df_token_lists[c].fillna(value='', inplace=True)

df_lem_strings = pd.DataFrame(li_lem_strings, columns=['lem quote'])

print()
print("")
print("df_lem_strings.head():")
print(df_lem_strings.head().to_string())
```

Output:

```
df_token_lists.head(5):
    0         1  2  3  4  5  6      7         8  9  10 11        12       13      14       15  16  17       18 19 20    21   22 23      24  25         26     27  28    29
0 hello everybody     it  s  me     rajani lamichhane      i am currently learning machine learning and deep learning      i would love to connect with likeminded people who share


df_lem_strings.head():

0 hello everybody - - - - - rajani lamichhane - - - currently learn machine learn - deep learn - - - love - connect - likeminded people - share - - interest - minei - presently enrol - - second year - comp
```

Title.4: Process results, find the most popular lemmatized words and group results by Part of Speech (POS)

Source code:

```
# Add counts
print("Group by lemmatized words, add count and sort:")
df_all_words = pd.DataFrame(li_tokens, columns=['token', 'stem', 'lem', 'pos'])
df_all_words['counts'] = df_all_words.groupby(['lem'])['lem'].transform('count')
df_all_words = df_all_words.sort_values(by=['counts', 'lem'], ascending=[False, True]).reset_index()

print("Get just the first row in each lemmatized group")
df_words = df_all_words.groupby('lem').first().sort_values(by='counts', ascending=False).reset_index()
print("df_words.head(10):")
print(df_words.head(10))
```

Output:

```
Group by lemmatized words, add count and sort:
Get just the first row in each lemmatized group
df_words.head(10):
          lem  index       token       stem pos  counts
0       learn      5    learning      learn  VB       6
1        also     24        also       also  RB       2
2        deep      8        deep       deep  NN       2
3       woman     42       women      women  NN       2
4  technology     23  technology  technolog  NN       2
5     machine      6     machine     machin  NN       2
6        year     20        year       year  NN       2
7        past     49        past       past  NN       1
8      person     59      person     person  NN       1
9   presently     17   presently    present  RB       1
```

Title.5: Top 10 words per Part Of Speech (POS)

Source code:

```
df_words = df_words[['lem', 'pos', 'counts']].head(200)
for v in POS_TYPES:
df_pos = df_words[df_words['pos'] == v]
print()
print("POS_TYPE:", v)
print(df_pos.head(10).to_string())
```

Output:

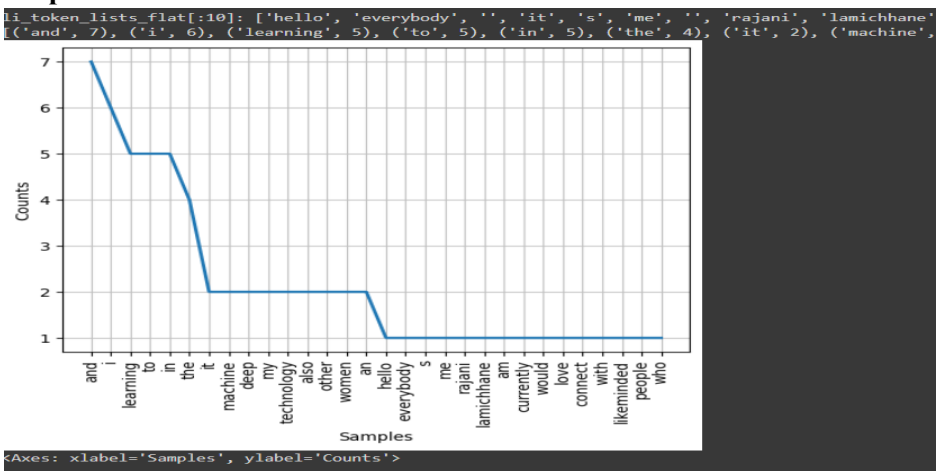
```
POS_TYPE: NN
          lem pos  counts
2        deep  NN       2
3       woman  NN       2
4  technology  NN       2
5     machine  NN       2
6        year  NN       2
7        past  NN       1
8      person  NN       1
10     rajani  NN       1
11    reading  NN       1
12   research  NN       1

POS_TYPE: JJ
           lem pos  counts
14      second  JJ       1
16  significant  JJ       1
26         big  JJ       1
30       nepal  JJ       1
48  beneficial  JJ       1

POS_TYPE: VB
          lem pos  counts
0       learn  VB       6
28  contribute  VB       1
34      depend  VB       1
37       enrol  VB       1
40   fascinate  VB       1
46  likeminded  VB       1

POS_TYPE: RB
         lem pos  counts
1       also  RB       2
9   presently  RB       1
```

Title.6: Sorted frequency plot for all words

Source code:

```
li_token_lists_flat = [y for x in li_token_lists for y in x]
# flatten the list of token lists to a single list
print("li_token_lists_flat[:10]:", li_token_lists_flat[:10])
di_freq = nltk.FreqDist(li_token_lists_flat)
del di_freq['']
li_freq_sorted = sorted(di_freq.items(), key=lambda x: x[1], reverse=True) # sorted list
print(li_freq_sorted)
di_freq.plot(30, cumulative=False)
```

Output:

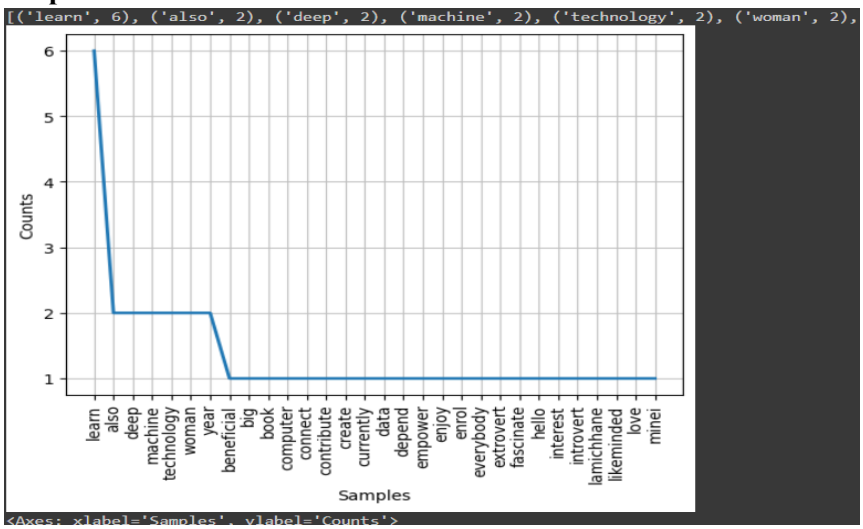


Title.7: Sorted frequency plot for Lemmatized words after removing stopwords.

Source code:

```
li_lem_words = df_all_words['lem'].tolist()
di_freq2 = nltk.FreqDist(li_lem_words)
li_freq_sorted2 = sorted(di_freq2.items(), key=lambda x: x[1], reverse=True) # sorted list
print(li_freq_sorted2)
di_freq2.plot(30, cumulative=False)
```

Output:



Title.8: Import the NLTK module and download the text resources needed for the examples.

Source code:

```
import nltk
# import all the resources for Natural Language Processing with Python
nltk.download("book")
```

Output:

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package dependency_treebank to
```

Title.9: Take a sentence and tokenize into words. Then apply a part-of-speech tagger.

Source code:

```
sentence = "My name is Rajani Lamichhane.”
tokens = nltk.word_tokenize(sentence)
print(tokens)
tagged = nltk.pos_tag(tokens)
print(tagged)
```

Output:

```
['My', 'name', 'is', 'Rajani', 'Lamichhane', '.']
[('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Rajani', 'NNP'), ('Lamichhane', 'NNP'), ('.', '.')]
```

Title.10: From the tagged words, identify the proper names.

Source code:

```
entities = nltk.chunk.ne_chunk(tagged)
print(entities)
```

Output:

```
(S My/PRP$ name/NN is/VBZ (PERSON Rajani/NNP Lamichhane/NNP) ./.)
```

Title.11: get texts for corpus analysis.

Source code:

```
%matplotlib inline
from nltk.book import *
```

Output:

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the material
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
```

Title.12: generate a key-word in context concordance.

Source code:

```
text1.concordance("monstrous")
```

Output:

```
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

Title.13: find words with similar concordance to a given word.

Source code:

```
import nltk
from nltk.book import text1, text2
# Print similar words for text1
print(text1)
text1.similar("monstrous")
# Print similar words for text2
print(text2)
text2.similar("monstrous")
```

Output:

```
<Text: Moby Dick by Herman Melville 1851>
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless
<Text: Sense and Sensibility by Jane Austen 1811>
very so exceedingly heartily a as good great extremely remarkably
sweet vast amazingly
```

Title.14: find contexts which are similar for the given words.

Source code:

```
import nltk
from nltk.book import text2
# Print common contexts for "monstrous" and "very" in text2
text2.common_contexts(["monstrous", "very"])
```

Output:

```
am_glad a_pretty a_lucky is_pretty be_glad
```

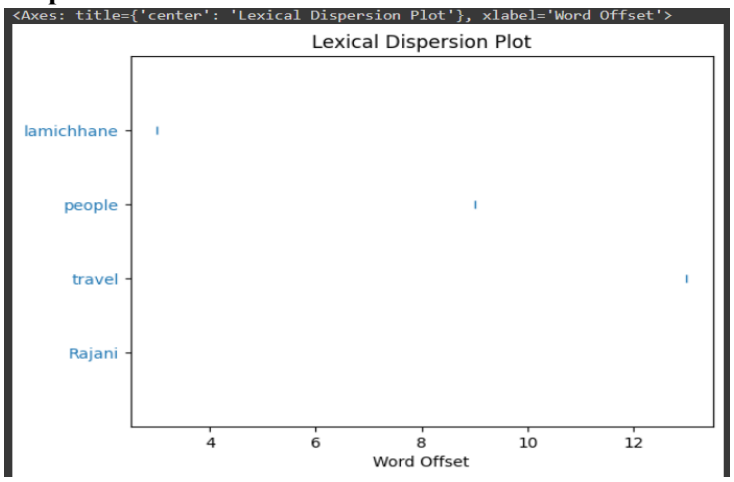
Title.15: plot where in the text certain words appear.

Source code:

```
import nltk
from nltk.draw import dispersion_plot
# Assuming you have your text stored in a variable called "text 4"
text = "My name is Rajani Lamichhane . I love to travel and meet new people." # Tokenize the text into individual words
tokens = nltk.word_tokenize(text)
# Create a dispersion plot
dispersion_plot(tokens, ["Rajani", "travel", "people", "lamichhane"])
```

Output:

```
<Axes: title={'center': 'Lexical Dispersion Plot'}, xlabel='Word Offset'>
```



Title.16:print the identity of a text, the length of the text and its vocabulary.

Source code:

```
print(text3)
print(len(text3))
print(sorted(set(text3)))
```

Output:

```
My name is Rajani Lamichhane. I love to travel and meet new people.
67
[' ', '.', 'I', 'L', 'M', 'R', 'a', 'c', 'd', 'e', 'h', 'i', 'j', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'v', 'w', 'y']
```

Title.17: print some statistics of word occurrence in the text.

Source code:

```
text = "My name is Rajani Lamichhane. I love to to travel and meet new people."
text1 = "I am currently pursuing BSC.CSIT."
text2 = "I am a passionate learner."
def lexical_diversity(text):
  return len(set(text)) / len(text)

def percentage(count, total):
  return 100 * count / total
print(lexical_diversity(text))
```

```
print(lexical_diversity(text2))
print(percentage(text1.count('a'), len(text1)))
```

Output:

```
0.3582089552238806
0.47058823529411764
2.9411764705882355
```