Self Study

Aim :

Recommender System is a system that seeks to predict or filter preferences according to the user’s choices.

Objective :

1. To comprehend the theoretical framework of a Recommender System.

2. To implement a basic Recommender System using Python.

3. To analyze and interpret the performance of the built Recommender System.

4. To explore the importance and applications of Recommender Systems in various domains.

Theory :

Recommender Systems are an essential component of modern information systems and are used in various applications, such as e-commerce platforms, content streaming services, social media, and more. The primary goal of a Recommender System is to predict the "rating" or "preference" that a user would give to an item. This is achieved by analyzing the user's past behaviour or preferences and providing suggestions or recommendations for items that the user might like.

There are mainly three types of Recommender Systems :

1. Content-based filtering: It recommends items similar to those a user has liked in the past.
2. Collaborative filtering: It predicts what a user might like based on the preferences of similar users.
3. Hybrid methods: These combine collaborative and content-based filtering to provide more accurate and effective recommendations.

In this lab, we will focus on building a simple content-based recommendation system using Python. We will use a dataset containing user preferences for various items to demonstrate the basic functionality of a Recommender System.

Materials Required :

a. Python 3.x
b. Jupyter Notebook or any Python IDE Pandas library

c. NumPy library Scikit-learn library

Experimental Procedure :

Step 1 : Data Collection and Preprocessing:

Import the necessary libraries.

Load the dataset into the environment.

Preprocess the dataset to make it suitable for the recommendation process.

Step 2 : Building the Recommendation Engine

Implement a content-based recommendation algorithm using appropriate similarity metrics. Construct a function that takes a user's preferences as input and recommends items based on those preferences.

Step 3 : Evaluation and Analysis

Evaluate the performance of the recommendation system using suitable metrics. Analyze the strengths and weaknesses of the implemented Recommender System.

Discuss the potential improvements that can be made to enhance the system's performance.

Step 4 : Discussion and Conclusion

Discuss the real-world applications of Recommender Systems. Summarize the key insights gained from the lab experiment.

Conclude with the significance of Recommender Systems in enhancing user experience and facilitating personalized recommendations.

Safety Precautions :

This lab does not involve any hazardous materials; hence, no specific safety precautions are required. However, standard coding practices must be followed to ensure data security and integrity.

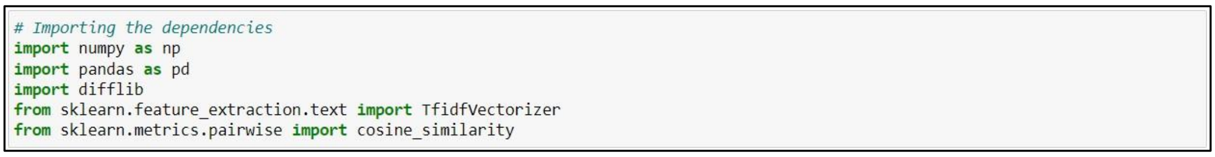
Conclusion :

This lab provides an insightful introduction to the concept of Recommender Systems and their practical implementation. By building a basic recommendation engine, you have gained a fundamental understanding of how these systems work and their significance in modern information systems. Understanding the strengths and limitations of such systems is crucial for developing more sophisticated and accurate recommendation engines for diverse applications in the future.

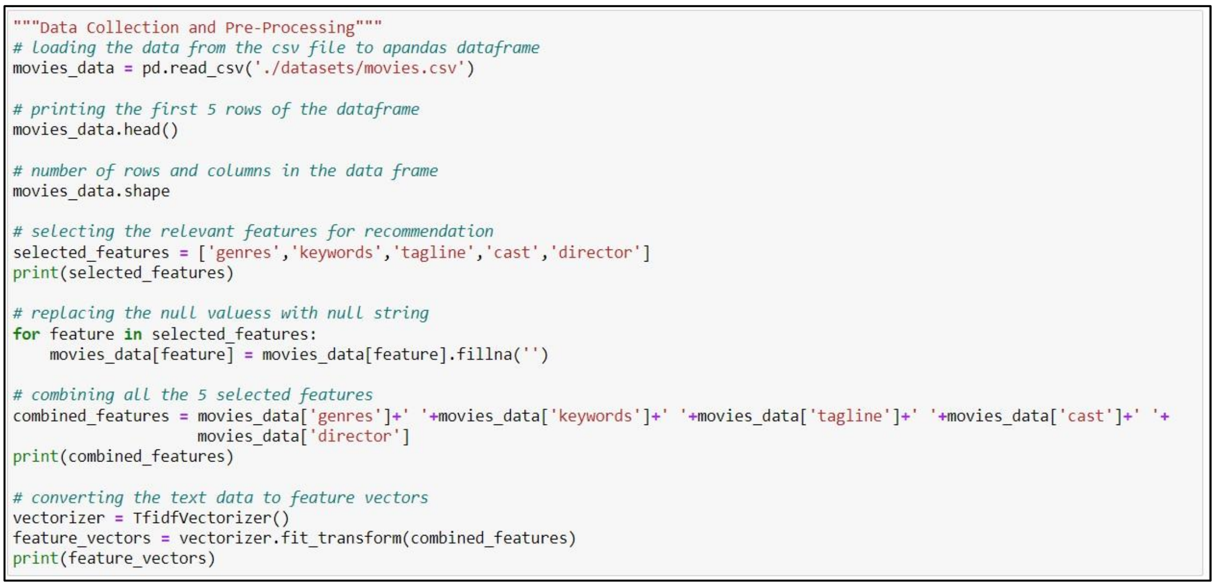
Name : Nikhil Sonone **Roll No :** 68

Academic Year – 2023 – 24 **Class and Department –** BE AI&DS

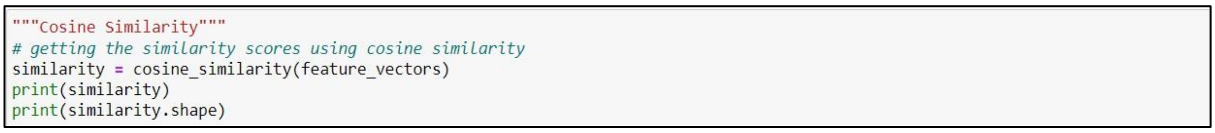
Code :



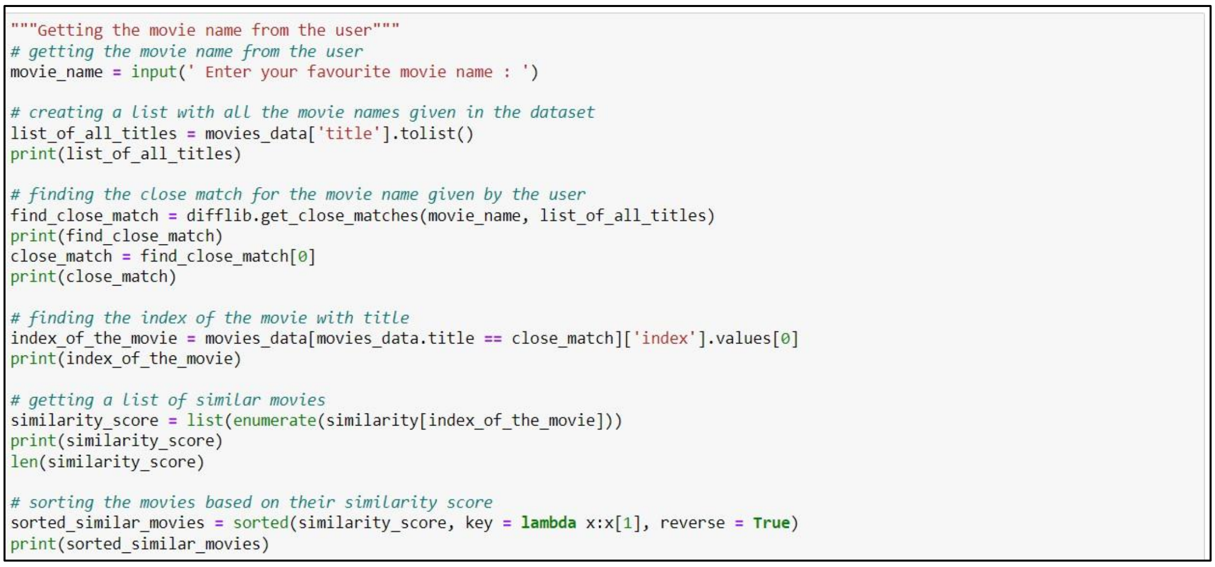
```python
# Importing the dependencies
import numpy as np
import pandas as pd
import difflib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```



```python
"""Data Collection and Pre-Processing"""
# loading the data from the csv file to apandas dataframe
movies_data = pd.read_csv('./datasets/movies.csv')

# printing the first 5 rows of the dataframe
movies_data.head()

# number of rows and columns in the data frame
movies_data.shape

# selecting the relevant features for recommendation
selected_features = ['genres','keywords','tagline','cast','director']
print(selected_features)

# replacing the null valuess with null string
for feature in selected_features:
    movies_data[feature] = movies_data[feature].fillna('')

# combining all the 5 selected features
combined_features = movies_data['genres']+' '+movies_data['keywords']+' '+movies_data['tagline']+' '+movies_data['cast']+' '+
                    movies_data['director']
print(combined_features)

# converting the text data to feature vectors
vectorizer = TfidfVectorizer()
feature_vectors = vectorizer.fit_transform(combined_features)
print(feature_vectors)
```



```python
"""Cosine Similarity"""
# getting the similarity scores using cosine similarity
similarity = cosine_similarity(feature_vectors)
print(similarity)
print(similarity.shape)
```



```python
"""Getting the movie name from the user"""
# getting the movie name from the user
movie_name = input(' Enter your favourite movie name : ')

# creating a list with all the movie names given in the dataset
list_of_all_titles = movies_data['title'].tolist()
print(list_of_all_titles)

# finding the close match for the movie name given by the user
find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
print(find_close_match)
close_match = find_close_match[0]
print(close_match)

# finding the index of the movie with title
index_of_the_movie = movies_data[movies_data.title == close_match]['index'].values[0]
print(index_of_the_movie)

# getting a list of similar movies
similarity_score = list(enumerate(similarity[index_of_the_movie]))
print(similarity_score)
len(similarity_score)

# sorting the movies based on their similarity score
sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)
print(sorted_similar_movies)
```

```python
# print the name of similar movies based on the index
print('Movies suggested for you : \n')
i = 1
for movie in sorted_similar_movies:
    index = movie[0]
    title_from_index = movies_data[movies_data.index==index]['title'].values[0]
    if i<30:
        print(f"{i}. {title_from_index}")
        i+=1
```

User Input :

Enter your favourite movie – Hulk

Output :

```
Movies suggested for you :

1. Hulk
2. The Helix... Loaded
3. House of Sand and Fog
4. Terminator Salvation
5. Northfork
6. Zodiac
7. Star Trek
8. Wolves
9. Milk
10. Stealth
11. Warrior
12. Gangster Squad
13. He's Just Not That Into You
14. Draft Day
15. George of the Jungle
16. The Age of Adaline
17. Parker
18. Antz
19. Star Trek: The Motion Picture
20. Capricorn One
21. Dark City
22. Iron Man 3
23. Cloverfield
24. Star Trek IV: The Voyage Home
25. The Time Traveler's Wife
26. Jurassic World
27. Spider-Man 3
28. The Legend of Zorro
29. Noah
```

Prerequisites 1

Aim :

To install NLTK for windows.

Objective :

To Study:

1. The usage of libraries or datasets of NLTK.

Theory :

Installing NLTK :

NLTK requires Python versions 3.7, 3.8, 3.9, 3.10 or 3.11.

For Windows users, it is strongly recommended that you go through this guide to install Python 3 successfully https://docs.pythonguide.org/starting/install3/win/#install3-windows

Setting up a Python Environment (Mac/Unix/Windows)

Please go through this guide to learn how to manage your virtual environment managers before you install NLTK, https://docs.python-guide.org/dev/virtualenvs/

Alternatively, you can use the Anaconda distribution installer that comes "batteries included" https://www.anaconda.com/distribution/

Mac/Unix

1. Install NLTK: run pip install --user -U nltk
2. Install Numpy (optional): run pip install --user -U numpy
3. Test installation: run python then type import nltk

For older versions of Python it might be necessary to install setup tools (see https://pypi.python.org/pypi/setuptools) and to install pip (sudo easy_install pip).

Windows

These instructions assume that you do not already have Python installed on your machine.

32-bit binary installation

1. Install Python 3.8: https://www.python.org/downloads/ (avoid the 64-bit versions)

2. Install Numpy (optional): https://numpy.org/install/
3. Install NLTK: https://pypi.python.org/pypi/nltk
4. Test installation: Start>Python38, then type import nltk

Installing NLTK Data

After installing the NLTK package, please do install the necessary datasets/models for specific functions to work. If you're unsure of which datasets/models you'll need, you can install the "popular" subset of NLTK data, on the command line type python -m nltk.downloader popular, or in the Python interpreter import nltk; nltk.download('popular')

Installing NLTK Data

NLTK comes with many corpora, toy grammars, trained models, etc. A complete list is posted at: https://www.nltk.org/nltk_data/

To install the data, first install NLTK (see https://www.nltk.org/install.html), then use NLTK's data downloader as described below.

Apart from individual data packages, you can download the entire collection (using "all"), or just the data required for the examples and exercises in the book (using "book"), or just the corpora and no grammars or trained models (using "all-corpora").

Interactive Installer

For central installation on a multi-user machine, do the following from an administrator account. Run the Python interpreter and type the commands:

```
>>> import nltk
>>> nltk.download()
```

A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory. For central installation, set this to C:\nltk_data (Windows), /usr/local/share/nltk_data (Mac),or /usr/share/nltk_data (Unix). Next, select the packages or collections you want to download.

If you did not install the data to one of the above central locations, you will need to set the NLTK_DATA environment variable to specify the location of the data. (On a Windows machine, right click on "My Computer" then select *Properties > Advanced > Environment Variables > User Variables > New...*)

Test that the data has been installed as follows. (This assumes you downloaded the Brown Corpus) :

```
>>> from nltk.corpus import brown
```

***>>>** brown.words()*

['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

Installing Via A Proxy Web Server

If your web connection uses a proxy server, you should specify the proxy address as follows. In the case of an authenticating proxy, specify a username and password. If the proxy is set to None then this function will attempt to detect the system proxy.

***>>>** nltk.set_proxy('http://proxy.example.com:3128', ('USERNAME', 'PASSWORD'))*

***>>>** nltk.download()*

Command Line Installation :

The downloader will search for an existing nltk_data directory to install NLTK data. If one does not exist it will attempt to create one in a central location (when using an administrator account) or otherwise in the user's filespace. If necessary, run the download command from an administrator account, or using sudo. The recommended system location is C:\nltk_data (Windows); /usr/local/share/nltk_data (Mac); and /usr/share/nltk_data (Unix). You can use the -d flag to specify a different location (but if you do this, be sure to set the NLTK_DATA environment variable accordingly).

Run the command python -m nltk.downloader all. To ensure central installation, run the command sudo python -m nltk.downloader -d /usr/local/share/nltk_data all.

Windows: Use the "Run..." option on the Start menu. Windows Vista users need to first turn on this option, using *Start -> Properties -> Customize* to check the box to activate the "Run..." option.

Test the Installation : Check that the user environment and privileges are set correctly by logging in to a user account, starting the Python interpreter, and accessing the Brown Corpus (see the previous section).

Manual Installation :

Create a folder nltk_data, e.g. C:\nltk_data, or /usr/local/share/nltk_data, and subfolders chunkers, grammars, misc, sentiment, taggers, corpora, help, models, stemmers, tokeni zers.

Download individual packages from https://www.nltk.org/nltk_data/ (see the "download" links). Unzip them to the appropriate subfolder. For example, the Brown Corpus, found at:

https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/corpora/brown.zip is to be unzipped to nltk_data/corpora/brown.

Set your NLTK_DATA environment variable to point to your top level nltk_data folder.

Conclusion :

Thus, we have implemented the installation of NLTK libraries.

Outcome :

Upon completion of this experiment, students will be able to :

Experiment level outcome (ELO1): Understand the installation of NLTK

Questions:

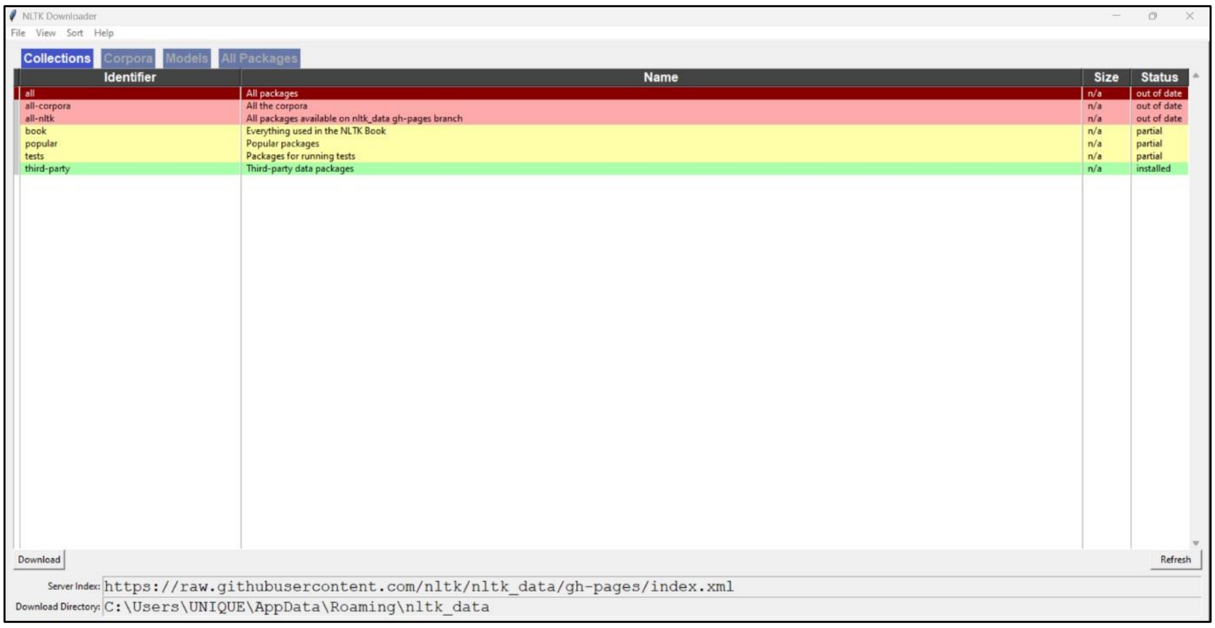
1. What is NLTK?
2. State some libraries of NLTK
3. State the steps of installation of NLTK on Windows

Name : Nikhil Sonone **Roll No :** 68

Academic Year – 2023 – 24 **Class and Department –** BE AI&DS

Downloading NLTK :

```
In [1]: import nltk
        nltk.download()

        showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

Out[1]: True
```



Identifier	Name	Size	Status
all	All packages	n/a	out of date
all-corpora	All the corpora	n/a	out of date
all-nltk	All packages available on nltk_data gh-pages branch	n/a	out of date
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	partial
tests	Packages for running tests	n/a	partial
third-party	Third-party data packages	n/a	installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: C:\Users\UNIQUE\AppData\Roaming\nltk_data

Checking NLTK Corpus Installation :

```
In [2]: from nltk.corpus import brown
        brown.words()

Out[2]: ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Prerequisites 2

Aim :

Implement simple programs with NLTK: Implement numpy array operations (any 5), named entity recognition.

Objective :

To Study:

1. Become familiar with the diverse libraries of NLTK.
2. Learn to import libraries for specific functions.

Theory :

What is NumPy ?

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

Why Use NumPy ?

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists ?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behaviour is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in ?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase ?

The source code for NumPy is located at this GitHub repository https://github.com/numpy/numpy

Array in Numpy :

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as ndarray. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

Creating a Numpy Array :

Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.

Steps to create Numpy Array :

1. Import the numpy package
2. Use np.array(list/tuple) to create array in numpy 1D/2D/N-D
3. Np.array(list) will return array variable to perform array operations on it.

Named Entity Recognition :

The named entity recognition (NER) is one of the most popular data preprocessing task. It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistently talked about or refer to in the text. NER is the form of NLP. At its core, NLP is just a two-step process, below are the two steps that are involved :

- Detecting the entities from the text
- Classifying them into different categories
- Some of the categories that are the most important architecture in NER such that:

- Person
- Organization
- Place/ location

Other common tasks include classifying of the following :

- date/time.
- expression
- Numeral measurement (money, percent, weight, etc)
- E-mail address

Ambiguity in NE :

- For a person, the category definition is intuitively quite clear, but for computers, there is some ambiguity in classification. Let's look at some ambiguous example:
  - England (Organisation) won the 2019 world cup vs The 2019 world cup happened in England(Location).
  - Washington(Location) is the capital of the US vs The first president of the US was Washington(Person).

Methods of NER :

- One way is to train the model for multi-class classification using different machine learning algorithms, but it requires a lot of labelling. In addition to labelling the model also requires a deep understanding of context to deal with the ambiguity of the sentences. This makes it a challenging task for a simple machine learning algorithm.
- Another way is that Conditional random field that is implemented by both NLP Speech Tagger and NLTK. It is a probabilistic model that can be used to model sequential data such as words. The CRF can capture a deep understanding of the context of the sentence. In this model, the input.

$$X = \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_T\}$$

$$p(y—x) = \frac{1}{z(\vec{x})} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{k} \omega_k f_k(y_t, y_{t-1}, \vec{x}_t)\right\}$$

- **Deep Learning Based NER:** deep learning NER is much more accurate than previous method, as it is capable to assemble words. This is due to the fact that it used a method called word embedding, that is capable of understanding the semantic and syntactic relationship between various words. It is also able to learn analyzes topic-specific as well as high level words automatically. This makes deep learning NER applicable for performing multiple tasks. Deep learning can do most of the repetitive work itself, hence researchers for example can use their time more efficiently.

Implementation :

In this implementation, we will perform Named Entity Recognition using two different frameworks: Spacy and NLTK. This code can be run on colab, however for visualization purpose. I recommend the local environment. We can install the following frameworks using pip install.

Follow the steps :

1. Import necessary modules and download packages.
2. Process the text and print named entities.
3. Then follow the tokenization steps.

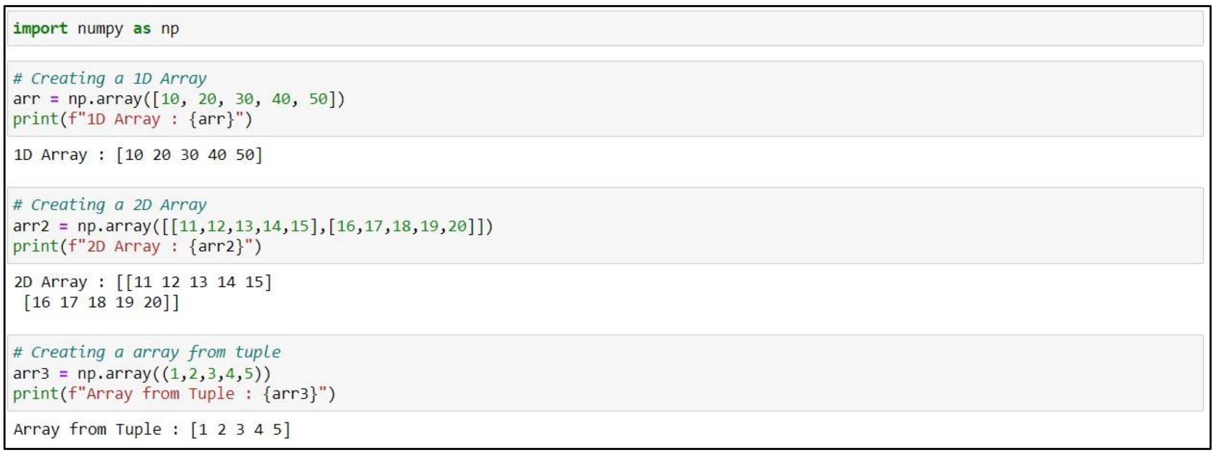
Conclusion :

Hence, we downloaded and used numpy package for creation and manipulation of array and also identified and classified key information from the text using Named Entity Recognition.

Name : Nikhil Sonone **Roll No :** 68

Academic Year – 2023 – 24 **Class and Department –** BE AI&DS

Numpy Array Creation :



```
import numpy as np
```

```
# Creating a 1D Array
arr = np.array([10, 20, 30, 40, 50])
print(f"1D Array : {arr}")
```

```
1D Array : [10 20 30 40 50]
```

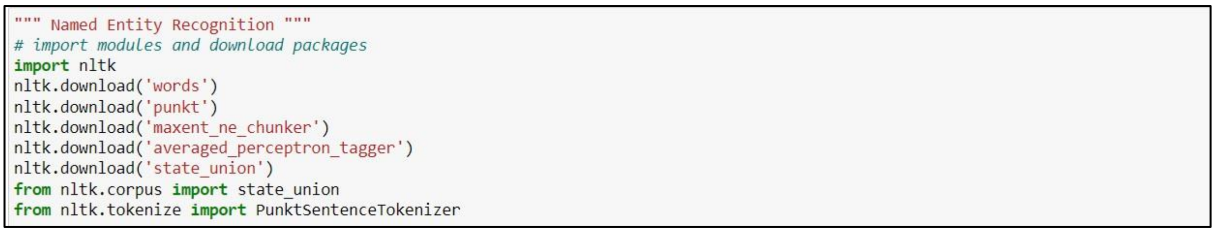
```
# Creating a 2D Array
arr2 = np.array([[11,12,13,14,15],[16,17,18,19,20]])
print(f"2D Array : {arr2}")
```

```
2D Array : [[11 12 13 14 15]
 [16 17 18 19 20]]
```

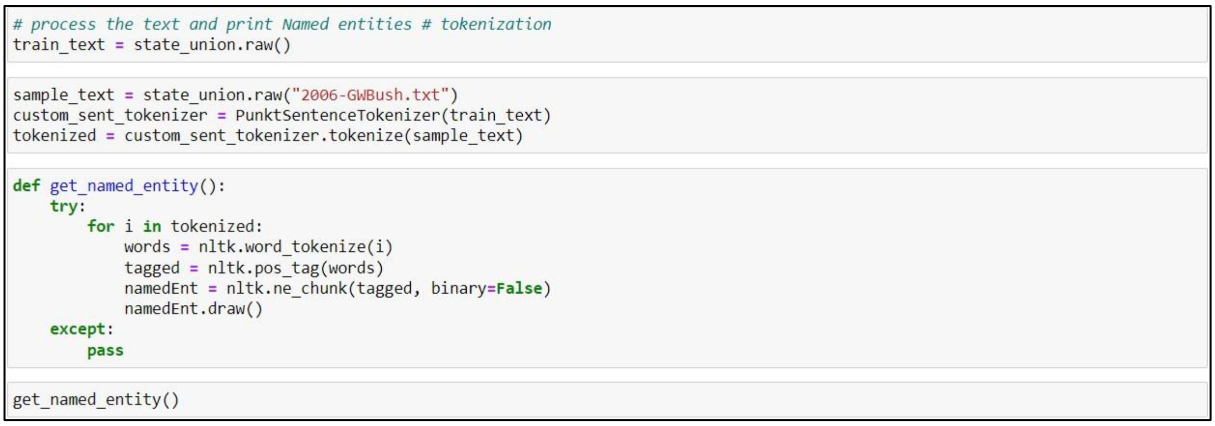
```
# Creating a array from tuple
arr3 = np.array((1,2,3,4,5))
print(f"Array from Tuple : {arr3}")
```

```
Array from Tuple : [1 2 3 4 5]
```

Named Entity Recognition :



```
""" Named Entity Recognition """
# import modules and download packages
import nltk
nltk.download('words')
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
nltk.download('state_union')
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer
```



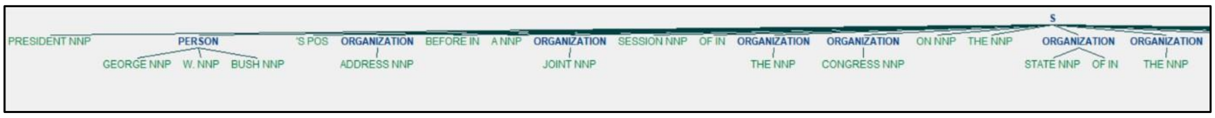
```
# process the text and print Named entities # tokenization
train_text = state_union.raw()
```

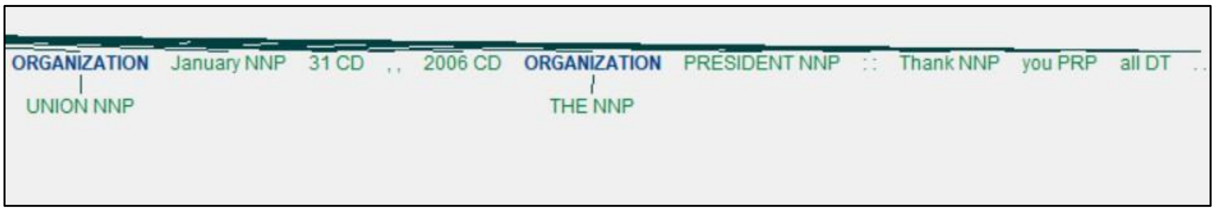
```
sample_text = state_union.raw("2006-GWBush.txt")
custom_sent_tokenizer = PunktSentenceTokenizer(train_text)
tokenized = custom_sent_tokenizer.tokenize(sample_text)
```

```
def get_named_entity():
    try:
        for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)
            namedEnt = nltk.ne_chunk(tagged, binary=False)
            namedEnt.draw()
    except:
        pass
```

```
get_named_entity()
```

Output :


S
PRESIDENT NNP
PERSON
GEORGE NNP
W. NNP
BUSH NNP
'S POS
ORGANIZATION
ADDRESS NNP
BEFORE IN
A NNP
ORGANIZATION
JOINT NNP
SESSION NNP
OF IN
ORGANIZATION
THE NNP
ORGANIZATION
CONGRESS NNP
ON NNP
THE NNP
ORGANIZATION
STATE NNP
OF IN
ORGANIZATION
THE NNP


ORGANIZATION
UNION NNP
January NNP
31 CD
, ,
2006 CD
ORGANIZATION
THE NNP
PRESIDENT NNP
: :
Thank NNP
you PRP
all DT
. .

Practical No. 01

Aim :

Write a program for pre-processing of a text document such as stop word removal, stemming.

Objective :

To Study :

1. Pre-processing of text documents
2. Removing Stop Words with NLTK
3. Performing the Stop words operations in a file

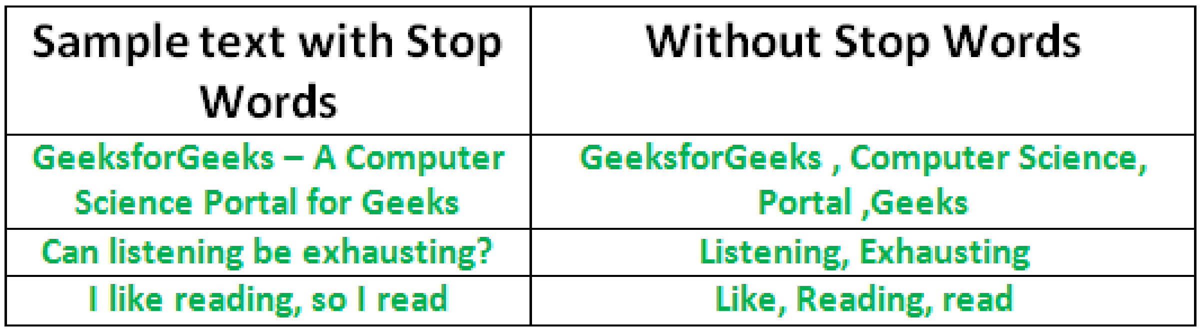
Theory :

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

What are Stop words?

Stop Words : A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory. home/pratima/nltk_data/corpora/stopwords are the directory address.(Do not forget to change your home directory name)



Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Removing stop words with NLTK :

a) The following program removes stop words from a piece of text :

1. Import “stopwords” module from “nlt.corpus “library.
2. Import “word_tokenize” module from “nltk.tokenize” library.
3. Get stopwords from the above package (usually English language).
4. First tokenize our sample text or sentence using “word_tokenize”.
5. Then converts the words in word_tokens to lower case and then checks whether they are present in stop_words or not.

b) Performing the Stopwords operations in a file :

1. Import all the necessary packages and module regarding stopwords removal and handling file in Python.
2. Get stopwords from the above package (usually English language).
3. “word_tokenize” accept only string, so first read the file, split the file data into comma separated words.
4. Then converts the words in word_tokens to lower case and then checks whether they are present in stop_words or not.
5. While checking each word as stop word or not, side by side create a new file and append those words in that file which are not stopwords in our Original file and named that file as formatted / output file.

This is how we are making our processed content more efficient by removing words that do not contribute to any future operations.

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words. How do we get these tokenized words? Well, tokenization involves breaking down the document into different words.

Stemming is a natural language processing technique that is used to reduce words to their base form, also known as the root form. The process of stemming is used to normalize text and make it easier to process. It is an important step in text pre-processing, and it is commonly used in information retrieval and text mining applications.

There are several different algorithms for stemming, including the Porter stemmer, Snowball stemmer, and the Lancaster stemmer. The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words. The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English. The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer.

Stemming can be useful for several natural language processing tasks such as text classification, information retrieval, and text summarization. However, stemming can also have some negative effects such as reducing the readability of the text, and it may not always produce the correct root form of a word.

Errors in Stemming :

There are mainly two errors in stemming –

1. over-stemming

2. under-stemming

Over-stemming occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false-positives. Over-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer produces a root form that is not a valid word or is not the correct root form of a word. This can happen when the stemmer is too aggressive in removing suffixes or when it does not consider the context of the word.

Over-stemming can lead to a loss of meaning and make the text less readable. For example, the word “arguing” may be stemmed to “argu,” which is not a valid word and does not convey the same meaning as the original word. Similarly, the word “running” may be stemmed to “run,” which is the base form of the word but it does not convey the meaning of the original word.

To avoid over-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing valid root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.

Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Under-stemming occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false-negatives. Under-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer does not produce the correct root form of a word or does not reduce a word to its base form. This can happen when the stemmer is not aggressive enough in removing suffixes or when it is not designed for the specific task or language.

Under-stemming can lead to a loss of information and make it more difficult to analyze text. For example, the word “arguing” and “argument” may be stemmed to “argu,” which does not convey the meaning of the original words. Similarly, the word “running” and “runner” may be stemmed to “run,” which is the base form of the word but it does not convey the meaning of the original words.

To avoid under-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing the correct root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.

Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Applications of stemming :

i. Stemming is used in information retrieval systems like search engines. It is used to determine domain vocabularies in domain analysis.
ii. To display search results by indexing while documents are evolving into numbers and to map documents to common subjects by stemming.
iii. Sentiment Analysis, which examines reviews and comments made by different users about anything, is frequently used for product analysis, such as for online retail stores. Before it is interpreted, stemming is accepted in the form of the text-preparation mean.
iv. A method of group analysis used on textual materials is called document clustering (also known as text clustering). Important uses of it include subject extraction, automatic document structuring, and quick information retrieval.
v. Fun Fact: Google search adopted a word stemming in 2003. Previously a search for “fish” would not have returned “fishing” or “fishes”.

Some Stemming algorithms are :

1. **Porter’s Stemmer Algorithm :**

It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.

Example : EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

Advantage : It produces the best output as compared to other stemmers and it has less error rate. Limitation: Morphological variants produced are not always real words.

2. **Lovins Stemmer :**

It is proposed by Lovins in 1968, that removes the longest suffix from a word then the word is recorded to convert this stem into valid words.

Example : sitting -> sitt -> sit

Advantage : It is fast and handles irregular plurals like 'teeth' and 'tooth' etc. Limitation: It is time consuming and frequently fails to form words from stem.

3. **Dawson Stemmer :**

It is an extension of Lovins stemmer in which suffixes are stored in the reversed order indexed by their length and last letter.

Advantage : It is fast in execution and covers more suffices. Limitation: It is very complex to implement.

4. **Krovetz Stemmer :**

It was proposed in 1993 by Robert Krovetz. Following are the steps :

1) Convert the plural form of a word to its singular form.

2) Convert the past tense of a word to its present tense and remove the suffix 'ing'. Example: 'children' -> 'child'

Advantage : It is light in nature and can be used as pre-stemmer for other stemmers. Limitation: It is inefficient in case of large documents.

Xerox Stemmer Example:

'children' -> 'child'

'understood' -> 'understand' 'whom' -> 'who' 'best' -> 'good'

5. **N-Gram Stemmer :**

An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in common.

Example : ‘INTRODUCTIONS’ for n=2 becomes: *I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*

Advantage : It is based on string comparisons and it is language dependent.

Limitation : It requires space to create and index the n-grams and it is not time efficient.

6. Snowball Stemmer :

When compared to the Porter Stemmer, the Snowball Stemmer can map non-English words too. Since it supports other languages the Snowball Stemmers can be called a multi-lingual stemmer. The Snowball stemmers are also imported from the nltk package. This stemmer is based on a programming language called ‘Snowball’ that processes small strings and is the most widely used stemmer. The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred toas Porter2 Stemmer. Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed.

7. Lancaster Stemmer :

The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. But they are not as efficient as Snowball Stemmers. The Lancaster stemmers save the rules externally and basically uses an iterative algorithm.

Some more example of stemming for root word "like" include :

-> "likes"

-> "liked"

-> "likely"

-> "liking"

Errors in Stemming :

There are mainly two errors in stemming

Overstemming occurs when two words are stemmed from the same root that are of different stems. Under-stemming occurs when two words are stemmed from the same root that is not of different stems.

Applications of stemming are :

i. Stemming is used in information retrieval systems like search engines. It is used to determine domain vocabularies in domain analysis.
ii. Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.

Below is the implementation of stemming words using NLTK :

a) Choose some words to be stemmed -
- Importing necessary modules
- Iterate through those words and then convert each word its original stem / form

b) Stemming words from sentences -
- Importing necessary modules
- Then tokenization of sample text / sentence / simply splitting sentence into individual word
- Iterating and Converting each word into its original form / stem

Conclusion :

The pre-processing of text data not only reduces the dataset size but also helps us to focus on only useful and relevant data so that the future model would have a large percentage of efficiency. With the help of pre-processing techniques like tokenization, stemming, lemmatization, removing stop- words, and part of speech tag we can remove all the irrelevant text from our dataset and make our dataset ready for further processing or model building.

Applications :

1. To eliminate unimportant words

2. To allow applications to focus on the important words

3. To drop common words

Conclusion :

Thus we have implemented pre-processing of a text document such as stop word removal, stemming.

Outcome :

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform pre-processing of a text document such as stop word removal, stemming using NLTK

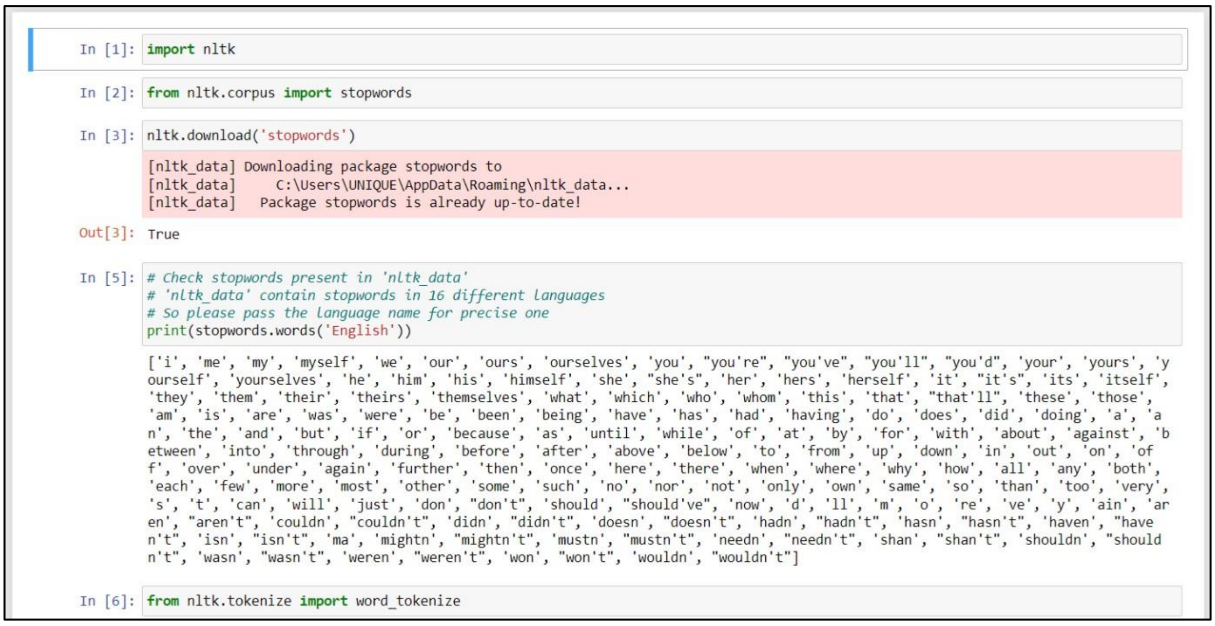
Questions :

1. What is stemming ?
2. Name any three Stemmer Algorithms.
3. What are the applications of stemming ?
4. What are Stopwords ?
5. Discuss about the errors in stemming? What are the two types of errors of Stemming?

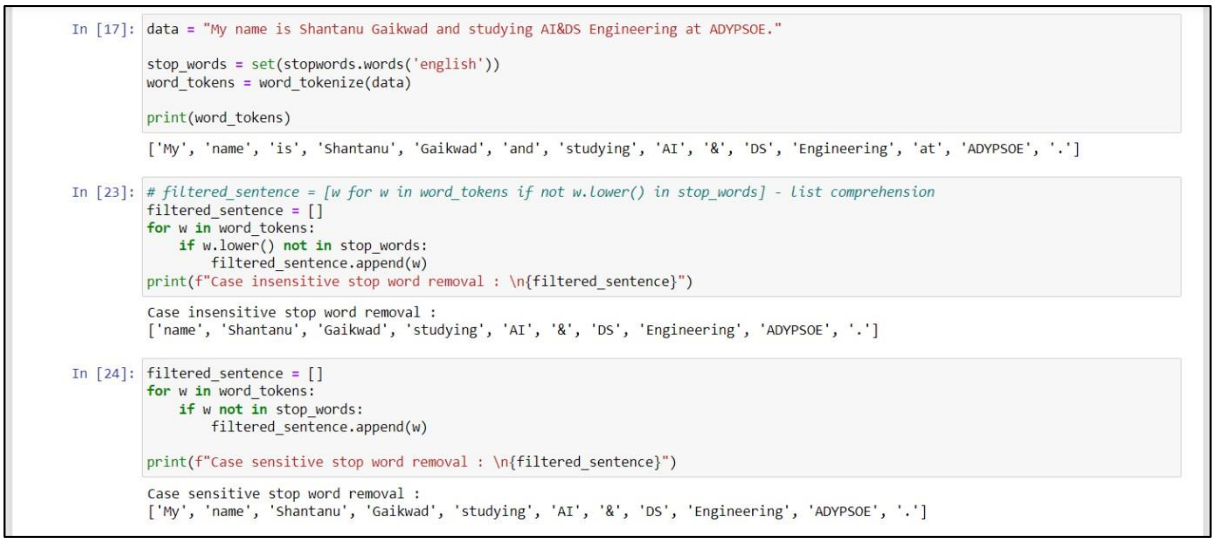
Name : Nikhil Sonone **Roll No :** 68

Academic Year – 2023 – 24 **Class and Department –** BE AI&DS

Stop Word Removal From Given Sample Text :

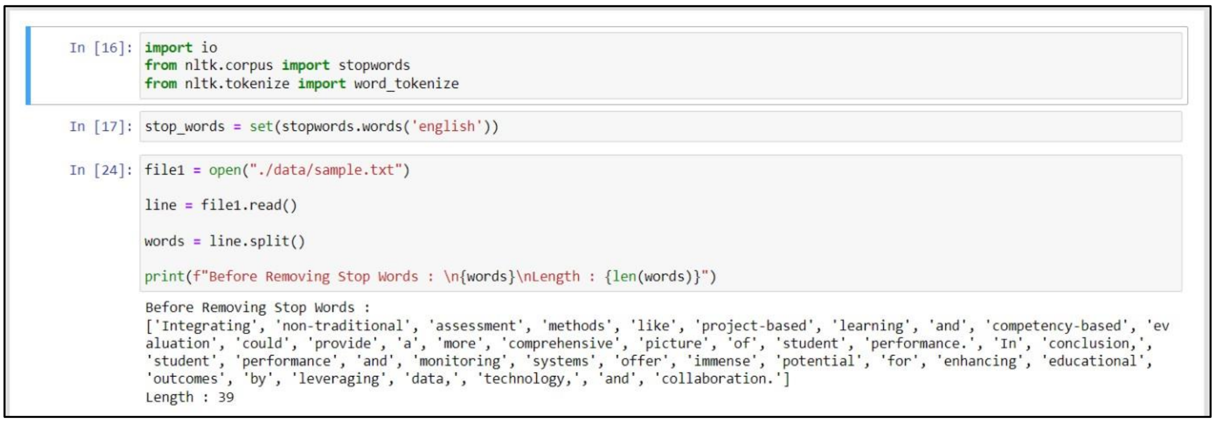


```
In [1]: import nltk

In [2]: from nltk.corpus import stopwords

In [3]: nltk.download('stopwords')

        [nltk_data] Downloading package stopwords to
        [nltk_data]     C:\Users\UNIQUE\AppData\Roaming\nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!

Out[3]: True

In [5]: # Check stopwords present in 'nltk_data'
        # 'nltk_data' contain stopwords in 16 different languages
        # So please pass the language name for precise one
        print(stopwords.words('English'))

        ['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"]

In [6]: from nltk.tokenize import word_tokenize
```

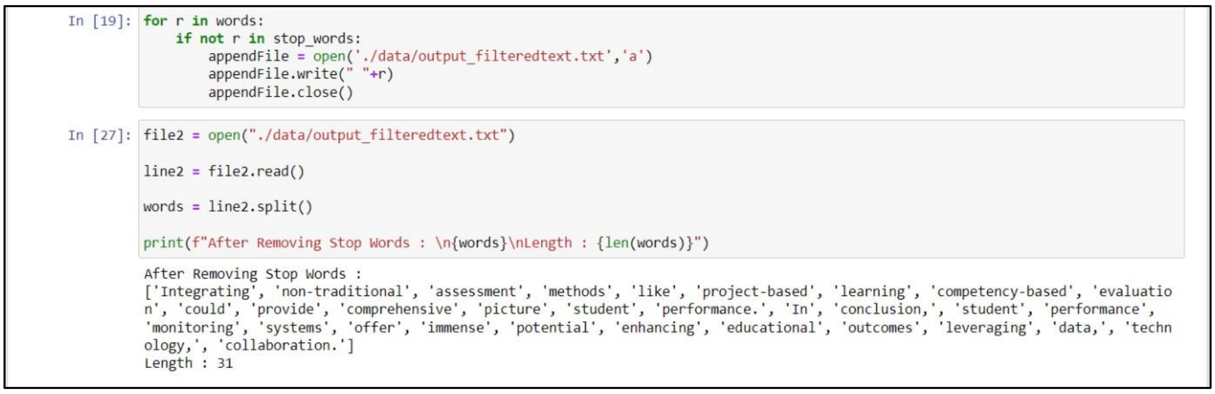


```
In [17]: data = "My name is Shantanu Gaikwad and studying AI&DS Engineering at ADYPSOE."

         stop_words = set(stopwords.words('english'))
         word_tokens = word_tokenize(data)

         print(word_tokens)

         ['My', 'name', 'is', 'Shantanu', 'Gaikwad', 'and', 'studying', 'AI', '&', 'DS', 'Engineering', 'at', 'ADYPSOE', '.']

In [23]: # filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words] - list comprehension
         filtered_sentence = []
         for w in word_tokens:
             if w.lower() not in stop_words:
                 filtered_sentence.append(w)
         print(f"Case insensitive stop word removal : \n{filtered_sentence}")

         Case insensitive stop word removal :
         ['name', 'Shantanu', 'Gaikwad', 'studying', 'AI', '&', 'DS', 'Engineering', 'ADYPSOE', '.']

In [24]: filtered_sentence = []
         for w in word_tokens:
             if w not in stop_words:
                 filtered_sentence.append(w)

         print(f"Case sensitive stop word removal : \n{filtered_sentence}")

         Case sensitive stop word removal :
         ['My', 'name', 'Shantanu', 'Gaikwad', 'studying', 'AI', '&', 'DS', 'Engineering', 'ADYPSOE', '.']
```

Stop Word Removal From Given File :

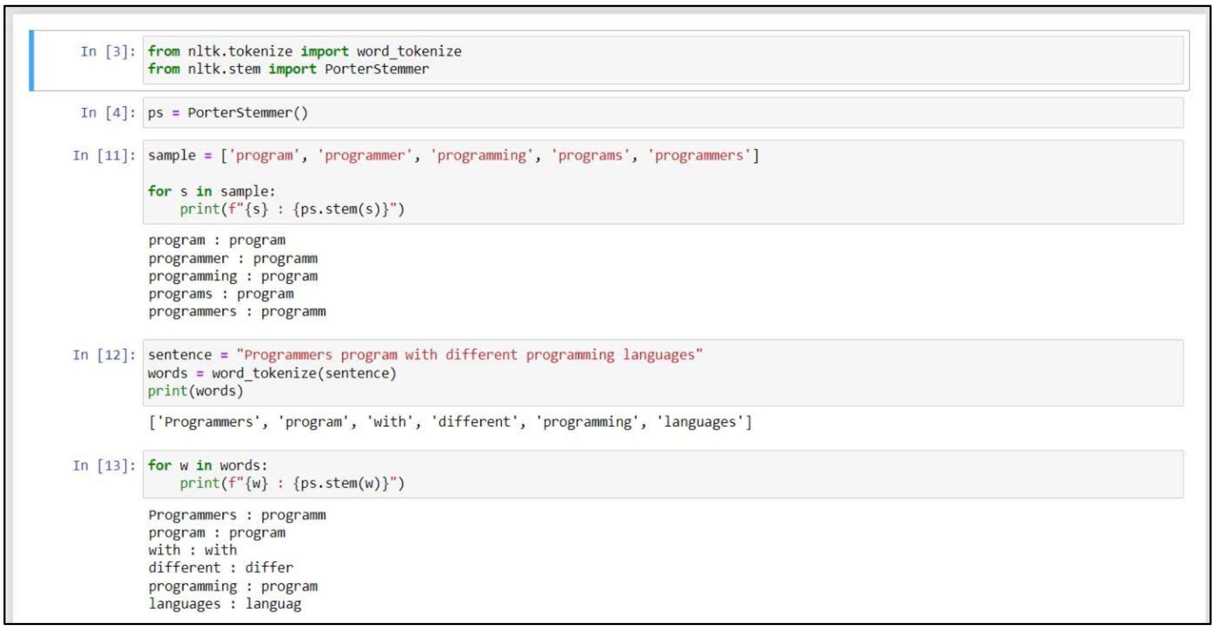


```
In [16]: import io
         from nltk.corpus import stopwords
         from nltk.tokenize import word_tokenize

In [17]: stop_words = set(stopwords.words('english'))

In [24]: file1 = open("./data/sample.txt")

         line = file1.read()

         words = line.split()

         print(f"Before Removing Stop Words : \n{words}\nLength : {len(words)}")

         Before Removing Stop Words :
         ['Integrating', 'non-traditional', 'assessment', 'methods', 'like', 'project-based', 'learning', 'and', 'competency-based', 'ev
         aluation', 'could', 'provide', 'a', 'more', 'comprehensive', 'picture', 'of', 'student', 'performance.', 'In', 'conclusion,',
         'student', 'performance', 'and', 'monitoring', 'systems', 'offer', 'immense', 'potential', 'for', 'enhancing', 'educational',
         'outcomes', 'by', 'leveraging', 'data,', 'technology,', 'and', 'collaboration.']
         Length : 39
```



```
In [19]: for r in words:
             if not r in stop_words:
                 appendFile = open('./data/output_filteredtext.txt','a')
                 appendFile.write(" "+r)
                 appendFile.close()

In [27]: file2 = open("./data/output_filteredtext.txt")

         line2 = file2.read()

         words = line2.split()

         print(f"After Removing Stop Words : \n{words}\nLength : {len(words)}")

         After Removing Stop Words :
         ['Integrating', 'non-traditional', 'assessment', 'methods', 'like', 'project-based', 'learning', 'competency-based', 'evaluatio
         n', 'could', 'provide', 'comprehensive', 'picture', 'student', 'performance.', 'In', 'conclusion,', 'student', 'performance',
         'monitoring', 'systems', 'offer', 'immense', 'potential', 'enhancing', 'educational', 'outcomes', 'leveraging', 'data,', 'techn
         ology,', 'collaboration.']
         Length : 31
```

Stemming :



```
In [3]: from nltk.tokenize import word_tokenize
        from nltk.stem import PorterStemmer

In [4]: ps = PorterStemmer()

In [11]: sample = ['program', 'programmer', 'programming', 'programs', 'programmers']

         for s in sample:
             print(f"{s} : {ps.stem(s)}")

         program : program
         programmer : programm
         programming : program
         programs : program
         programmers : programm

In [12]: sentence = "Programmers program with different programming languages"
         words = word_tokenize(sentence)
         print(words)

         ['Programmers', 'program', 'with', 'different', 'programming', 'languages']

In [13]: for w in words:
             print(f"{w} : {ps.stem(w)}")

         Programmers : programm
         program : program
         with : with
         different : differ
         programming : program
         languages : languag
```

Practical No. 02

Aim :

Implement a program for retrieval of documents using inverted files.

Objective :

To Study :

1. To efficiently retrieve documents or web pages containing a specific term or set of terms.
2. To retrieve documents using inverted files.

Theory :

An Inverted Index is a data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms. In an inverted index, the index is organized by terms (words), and each term points to a list of documents or web pages that contain that term.

Inverted indexes are widely used in search engines, database systems, and other applications where efficient text search is required. They are especially useful for large collections of documents, where searching through all the documents would be prohibitively slow.

An inverted index is an index data structure storing a mapping from content, such as words or number s, to its locations in a document or a set of documents. In simple words, it is a hashmap - like data structure that directs you from a word to a document or a web page.

To search for documents containing a particular term or set of terms, the search engine queries the inverted index for those terms and retrieves the list of documents associated with each term. The search engine can then use this information to rank the documents based on relevance to the query and present them to the user in order of importance.

There are two types of inverted indexes :

1. **Record-Level Inverted Index :** Record Level Inverted Index contains a list of references to documents for each word.
2. **Word-Level Inverted Index :** Word Level Inverted Index additionally contains the positions of each word within a document. The latter form offers more functionality but needs more processing power and space to be created.

Suppose we want to search the texts ***"hello everyone," "this article is based on an inverted index, " and "which is hashmap-like data structure".*** If we index by (text, word within the text), the index with a location in the text is :

hello (1, 1)

everyone (1, 2)

this (2, 1)

article (2, 2)

is (2, 3); (3, 2)

based (2, 4)

on (2, 5)

inverted (2, 6)

index (2, 7)

which (3, 1)

hashmap (3, 3)

like (3, 4)

data (3, 5)

structure (3, 6)

The word "hello" is in document 1 ("hello everyone") starting at word 1, so has an entry (1, 1), and the word "is" is in documents 2 and 3 at '3rd' and '2nd' positions respectively (here position is based on the word).The index may have weights, frequencies, or other indicators.

Steps to Build an Inverted Index :

1. **Fetch the Document :** Removing of Stop Words: Stop words are the most occurring and useless words in documents like "I", "the", "we", "is", and "an".
2. **Stemming of Root Word :** Whenever I want to search for "cat", I want to see a document that has information about it. But the word present in the document is called "cats" or "catty" instead of "cat". To relate both words, I'l chop some part of every word I read so that I could get the "root word". There are standard tools for performing this like "Porter's Stemmer".
3. **Record Document IDs :** If the word is already present add a reference of the document to index else creates a new entry. Add additional information like the frequency of the word, location of the word, etc.

Example :

Words	Document
ant	doc1
demo	doc2
world	doc1, doc2

Implementing Inverted Index :

1. Define the documents
2. Tokenize the documents
   - Convert each document to lowercase and split it into words
   - Combine the tokens into a list of unique terms
3. Build the inverted index
   - Create an empty dictionary to store the inverted index
   - For each term, find the documents that contain it
4. Print the inverted index

Advantages of Inverted Index :

1. The inverted index is to allow fast full-text searches, at a cost of increased processing when a document is added to the database.
2. It is easy to develop.
3. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

Disadvantages of Inverted Index :

1. Large storage overhead and high maintenance costs on updating, deleting, and inserting.
2. Instead of retrieving the data in decreasing order of expected usefulness, the records are retrieved in the order in which they occur in the inverted lists.

Features of Inverted Indexes :

1. **Efficient search :** Inverted indexes allow for efficient searching of large volumes of text- based data. By indexing every term in every document, the index can quickly identify all documents that contain a given search term or phrase, significantly reducing search time.

2. **Fast updates :** Inverted indexes can be updated quickly and efficiently as new content is added to the system. This allows for near-real-time indexing and searching for new content.
3. **Flexibility :** Inverted indexes can be customized to suit the needs of different types of information retrieval systems. For example, they can be configured to handle different types of queries, such as Boolean queries or proximity queries.
4. **Compression :** Inverted indexes can be compressed to reduce storage requirements. Various techniques such as delta encoding, gamma encoding, variable byte encoding, etc. can be used to compress the posting list efficiently.
5. **Support for stemming and synonym expansion :** Inverted indexes can be configured to support stemming and synonym expansion, which can improve the accuracy and relevance of search results. Stemming is the process of reducing words to their base or root form, while synonym expansion involves mapping different words that have similar meanings to a common term.
6. **Support for multiple languages :** Inverted indexes can support multiple languages, allowing users to search for content in different languages using the same system.

Applications :

1. Allow fast search for statistics related to the distinct words found in a text.

2. To allow fast full-text searches, at a cost of increased processing when a document is added to the database.

Input:

Example : Consider the following documents. Document 1 : The quick brown fox jumped over the lazy dog. Document 2 : The lazy dog slept in the sun.

Create an inverted index for these documents:

Output:

We first tokenize the documents into terms, as follows :

Document 1 : The, quick, brown, fox, jumped, over, the lazy, dog.

Document 2 : The, lazy, dog, slept, in, the, sun.

Next, we create an index of the terms, where each term points to a list of documents that contain that term, as follows.

The -> Document 1, Document 2 Quick -> Document 1

Brown -> Document 1 Fox-> Document 1 Jumped -> Document 1 Over-> Document 1

Lazy -> Document 1, Document 2

Dog -> Document 1, Document 2 Slept -> Document 2

In -> Document 2 Sun -> Document 2

Conclusion : This way, Implemented a program for inverted files.

Outcome :

Upon completion of this experiment, students will be able to :

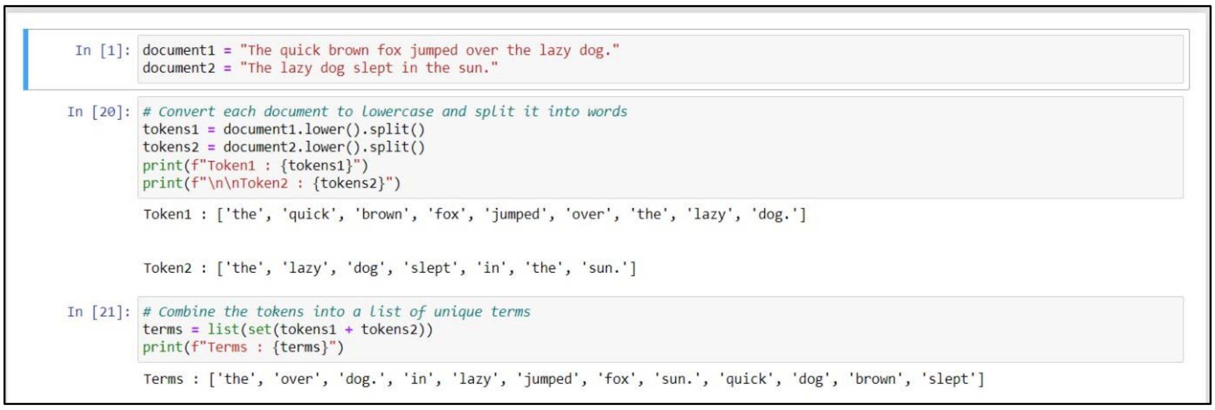
Experiment level outcome (ELO1): Perform how to retrieve documents using inverted files

Questions :

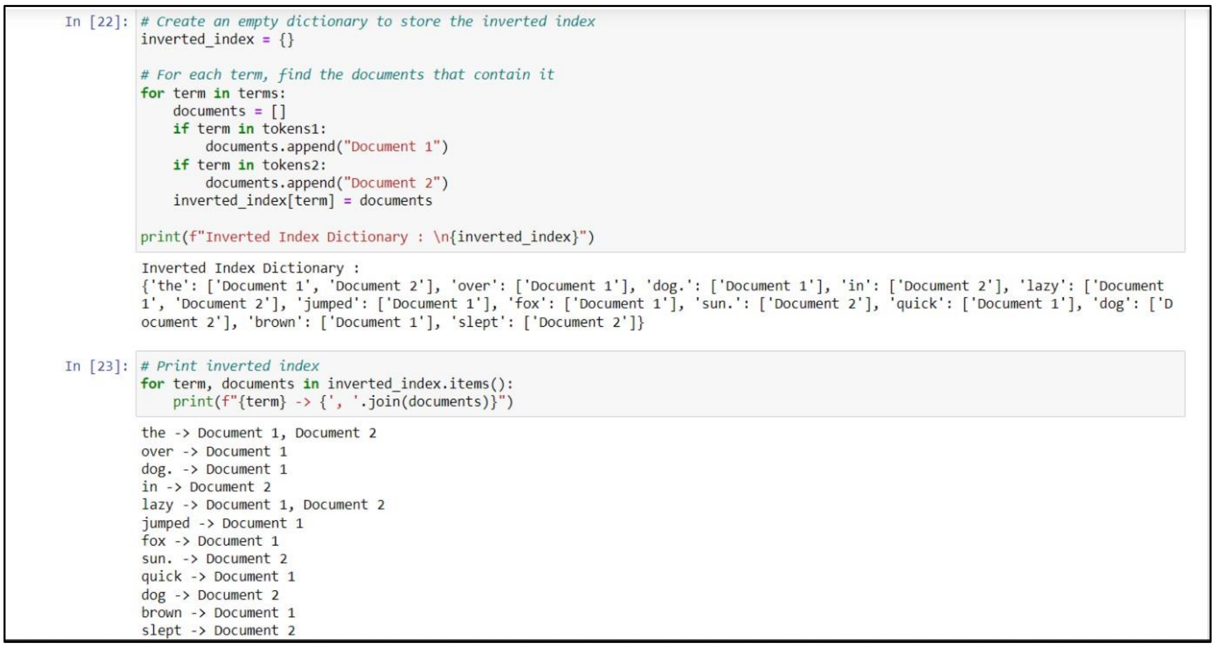
1. Why is it called inverted index ?
2. What are the two types of inverted indexes ?
3. What are the steps to build an inverted index ?
4. Enlist any five features of inverted index ?
5. What are the advantages and disadvantages of inverted index ?

Name : Nikhil Sonone **Roll No :** 68

Academic Year – 2023 – 24 **Class and Department –** BE AI&DS



```
In [1]: document1 = "The quick brown fox jumped over the lazy dog."
        document2 = "The lazy dog slept in the sun."

In [20]: # Convert each document to lowercase and split it into words
         tokens1 = document1.lower().split()
         tokens2 = document2.lower().split()
         print(f"Token1 : {tokens1}")
         print(f"\n\nToken2 : {tokens2}")

         Token1 : ['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog.']


         Token2 : ['the', 'lazy', 'dog', 'slept', 'in', 'the', 'sun.']

In [21]: # Combine the tokens into a list of unique terms
         terms = list(set(tokens1 + tokens2))
         print(f"Terms : {terms}")

         Terms : ['the', 'over', 'dog.', 'in', 'lazy', 'jumped', 'fox', 'sun.', 'quick', 'dog', 'brown', 'slept']
```



```
In [22]: # Create an empty dictionary to store the inverted index
         inverted_index = {}

         # For each term, find the documents that contain it
         for term in terms:
             documents = []
             if term in tokens1:
                 documents.append("Document 1")
             if term in tokens2:
                 documents.append("Document 2")
             inverted_index[term] = documents

         print(f"Inverted Index Dictionary : \n{inverted_index}")

         Inverted Index Dictionary :
         {'the': ['Document 1', 'Document 2'], 'over': ['Document 1'], 'dog.': ['Document 1'], 'in': ['Document 2'], 'lazy': ['Document
         1', 'Document 2'], 'jumped': ['Document 1'], 'fox': ['Document 1'], 'sun.': ['Document 2'], 'quick': ['Document 1'], 'dog': ['D
         ocument 2'], 'brown': ['Document 1'], 'slept': ['Document 2']}

In [23]: # Print inverted index
         for term, documents in inverted_index.items():
             print(f"{term} -> {', '.join(documents)}")

         the -> Document 1, Document 2
         over -> Document 1
         dog. -> Document 1
         in -> Document 2
         lazy -> Document 1, Document 2
         jumped -> Document 1
         fox -> Document 1
         sun. -> Document 2
         quick -> Document 1
         dog -> Document 2
         brown -> Document 1
         slept -> Document 2
```

Practical No. 03

Aim :

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set (You can use Java/Python ML library classes/API).

Objective :

To Study :

1. To construct a Bayesian network considering medical data.

Theory :

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditiona l dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts : a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.

The corresponding directed acyclic graph is depicted in below figure :

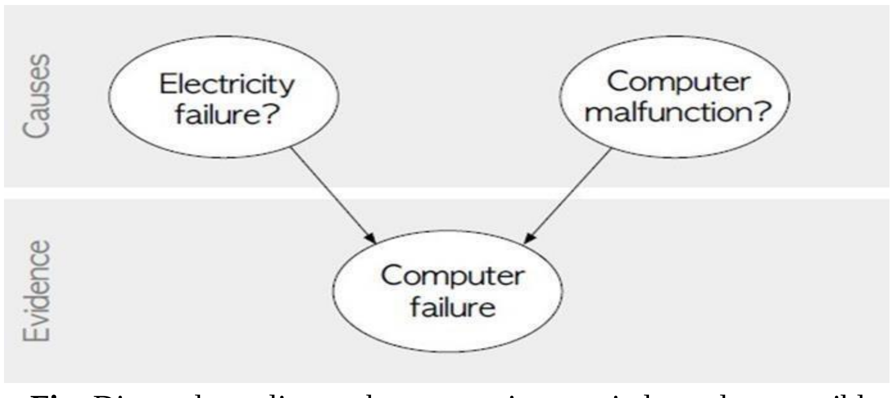


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Data Set :

Title : Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database :	0	1	2	3	4	Total
Cleveland :	164	55	36	35	13	303

Attribute Information :

01. age : age in years

02. sex : sex (1 = male; 0 = female)

03. cp : chest pain type

    - Value 1 : typical angina
    - Value 2 : atypical angina
    - Value 3 : non-anginal pain
    - Value 4 : asymptomatic

04. trestbps : resting blood pressure (in mm Hg on admission to the hospital)

05. chol : serum cholestoral in mg/dl

06. fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

07. restecg : resting electrocardiographic results

    - Value 0 : normal
    - Value 1 : having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2 : showing probable or definite left ventricular hypertrophy by Estes'criteria

08. thalach : maximum heart rate achieved

09. exang : exercise induced angina (1 = yes; 0 = no)

10. oldpeak : ST depression induced by exercise relative to rest

11. slope : the slope of the peak exercise ST segment

- Value 1 : upsloping
- Value 2 : flat
- Value 3 : downsloping

12. ca : number of major vessels (0-3) colored by flourosopy

13. tha l: 3 = normal; 6 = fixed defect; 7 = reversable defect

14. Heartdisease : It is integer valued from 0 (no presence) to 4. Diagnosis of heart disease(angiographic disease status).

age	sex	cp	trest bps	chol	fbs	reste cg	thalach	exang	oldpeak	slop e	ca	thal	Heart disease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program / Algorithm :

1. Import the require modules and packages
2. Read Cleveland Heart Disease data
3. Display the data
4. Model Bayesian Network
5. Learning CPDs using Maximum Likelihood Estimators
6. Inferencing with Bayesian Network
7. Computing the Probability of HeartDisease given Age
8. Computing the Probability of HeartDisease given cholesterol

Applications :

1. Medicine – Disease Diagnostics, Biomonitoring

2. Reasoning

3. Causal modeling

4. Decision making under uncertainty

5. Anomaly detection

Conclusion :

Thus we have constructed a Bayesian network considering medical data.

Outcome :

Upon completion of this experiment, students will be able to :

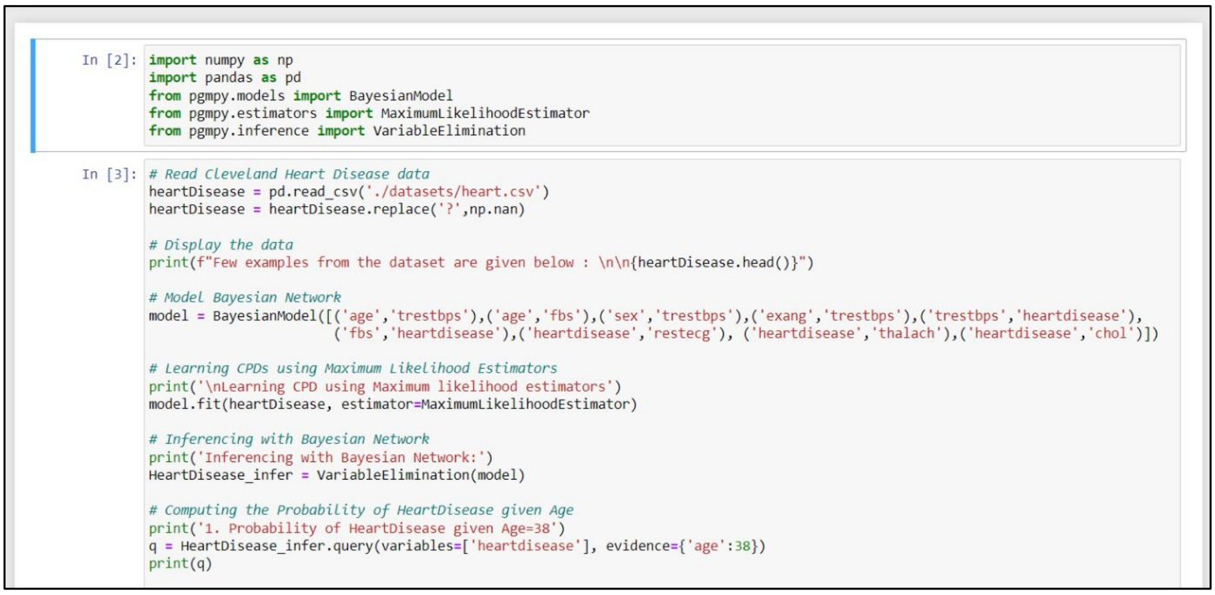
Experiment level outcome (ELO1) : After completion of this assignment students are able to understand how to construct a Bayesian network considering medical data.

Questions :

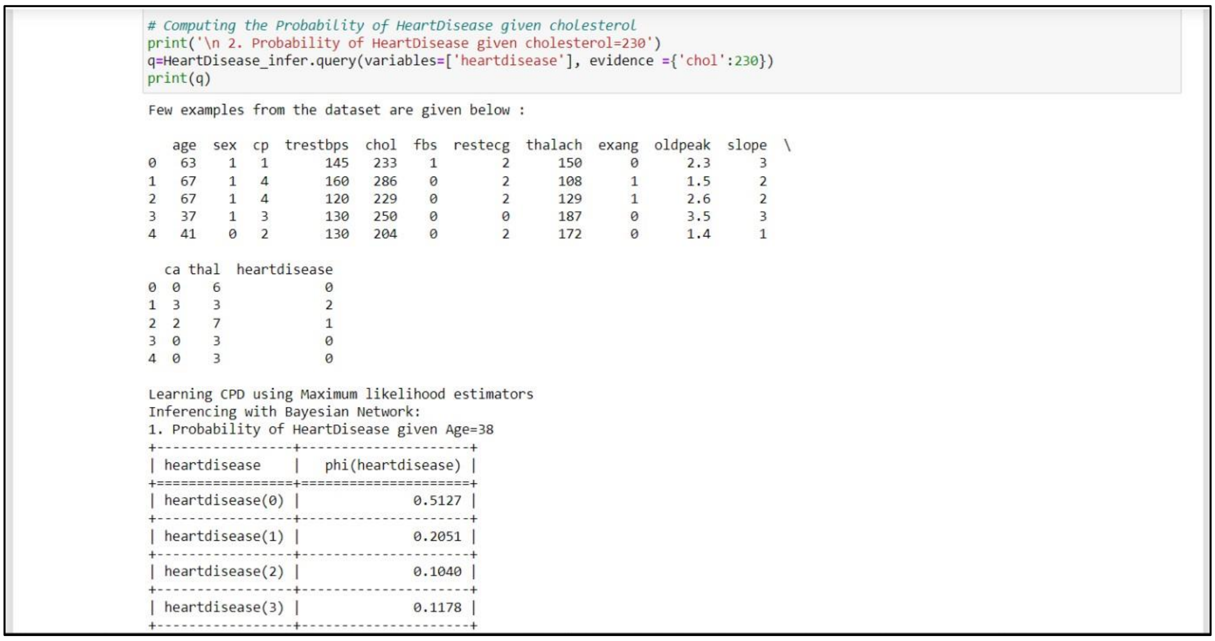
1. List any 10 real world applications of Bayesian Network.
2. What is Bayesian network used to represent?
3. What are turbo codes? How is it related to Bayesian Network?
4. What is a Bayesian Network?

Name : Nikhil Sonone **Roll No :** 68

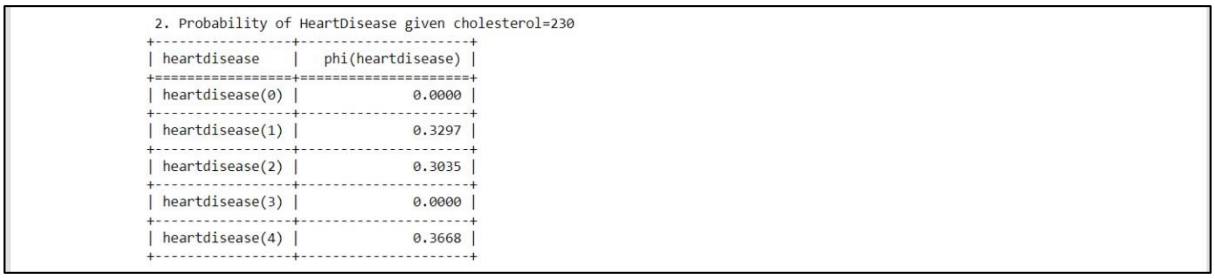
Academic Year – 2023 – 24 **Class and Department –** BE AI&DS



```
In [2]: import numpy as np
        import pandas as pd
        from pgmpy.models import BayesianModel
        from pgmpy.estimators import MaximumLikelihoodEstimator
        from pgmpy.inference import VariableElimination

In [3]: # Read Cleveland Heart Disease data
        heartDisease = pd.read_csv('./datasets/heart.csv')
        heartDisease = heartDisease.replace('?',np.nan)

        # Display the data
        print(f"Few examples from the dataset are given below : \n\n{heartDisease.head()}")

        # Model Bayesian Network
        model = BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),('trestbps','heartdisease'),
                               ('fbs','heartdisease'),('heartdisease','restecg'), ('heartdisease','thalach'),('heartdisease','chol')])

        # Learning CPDs using Maximum Likelihood Estimators
        print('\nLearning CPD using Maximum likelihood estimators')
        model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

        # Inferencing with Bayesian Network
        print('Inferencing with Bayesian Network:')
        HeartDisease_infer = VariableElimination(model)

        # Computing the Probability of HeartDisease given Age
        print('1. Probability of HeartDisease given Age=38')
        q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age':38})
        print(q)
```



```
        # Computing the Probability of HeartDisease given cholesterol
        print('\n 2. Probability of HeartDisease given cholesterol=230')
        q=HeartDisease_infer.query(variables=['heartdisease'], evidence ={'chol':230})
        print(q)

Few examples from the dataset are given below :

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0

Learning CPD using Maximum likelihood estimators
Inferencing with Bayesian Network:
1. Probability of HeartDisease given Age=38
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.5127 |
+-----------------+---------------------+
| heartdisease(1) |              0.2051 |
+-----------------+---------------------+
| heartdisease(2) |              0.1040 |
+-----------------+---------------------+
| heartdisease(3) |              0.1178 |
+-----------------+---------------------+
```



```
 2. Probability of HeartDisease given cholesterol=230
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.0000 |
+-----------------+---------------------+
| heartdisease(1) |              0.3297 |
+-----------------+---------------------+
| heartdisease(2) |              0.3035 |
+-----------------+---------------------+
| heartdisease(3) |              0.0000 |
+-----------------+---------------------+
| heartdisease(4) |              0.3668 |
+-----------------+---------------------+
```

Practical No. 04

Aim :

Implement Agglomerative hierarchical clustering algorithm using appropriate dataset

Objective :

To study :

1. Hierarchical Clustering Analysis

2. Implementation of agglomerative hierarchical clustering algorithm using appropriate dataset

Theory :

In data mining and statistics, hierarchical clustering analysis is a method of clustering analysis that seeks to build a hierarchy of clusters i.e. tree-type structure based on the hierarchy.

In machine learning, clustering is the unsupervised learning technique that groups the data based on similarity between the set of data. There are different-different types of clustering algorithms in machine learning. Connectivity-based clustering: This type of clustering algorithm builds the cluster based on the connectivity between the data points. Example : Hierarchical clustering.

Centroid-based clustering : This type of clustering algorithm forms around the centroids of the data points. Example: K-Means clustering, K-Mode clustering

Distribution-based clustering : This type of clustering algorithm is modeled using statistical distributions. It assumes that the data points in a cluster are generated from a particular probability distribution, and the algorithm aims to estimate the parameters of the distribution to group similar data points into clusters Example: Gaussian Mixture Models (GMM)

Density-based clustering : This type of clustering algorithm groups together data points that are in high-density concentrations and separates points in low-concentrations regions. The basic idea s that it identifies regions in the data space that have a high density of data points and groups those points together into clusters. Example: DBSCAN(Density-Based Spatial Clustering of Applications with Noise)

Hierarchical Clustering :

Hierarchical clustering is a connectivity-based clustering model that groups the data points together that are close to each other based on the measure of similarity or distance. The assumption is that datapoints that are close to each other are more similar or related than data points that are farther apart.

A dendrogram, a tree-like figure produced by hierarchical clustering, depicts the hierarchical relationships between groups. Individual data points are located at the bottom of the dendrogram, while the largest clusters, which include all the data points, are located at the top. In order to generate different numbers of clusters, the dendrogram can be sliced at various heights.

The dendrogram is created by iteratively merging or splitting clusters based on a measure of similarity or distance between data points. Clusters are divided or merged repeatedly until all data points are contained within a single cluster, or until the predetermined number of clusters is attained.

We can look at the dendrogram and measure the height at which the branches of the dendrogram form distinct clusters to calculate the ideal number of clusters. The dendrogram can be sliced at this height to determine the number of clusters.

Types of Hierarchical Clustering :

Basically, there are two types of hierarchical Clustering:

1. Agglomerative Clustering
2. Divisive clustering

Hierarchical Agglomerative Clustering :

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.

Agglomerative Clustering is one of the most common hierarchical clustering techniques. Dataset –Credit Card Dataset.

Assumption : The clustering technique assumes that each data point is similar enough to the other data points that the data at the starting can be assumed to be clustered in 1 cluster.

Algorithm :

1. Importing the required libraries
2. Loading and Cleaning the data
   - Changing the working location to the location of the CSV file
   - Dropping the CUST_ID column from the data
   - Handling the missing values
3. Preprocessing the data
   - Scaling the data so that all the features become comparable
   - Normalizing the data so that the data approximately follows a Gaussian distribution
   - Converting the numpy array into a pandas
4. Reducing the dimensionality of the Data
5. Visualizing the working of the Dendrograms
6. Building and Visualizing the different clustering models for different values of k
   - Visualizing the clustering
   - Appending the silhouette scores of the different models to the list
   - Plotting a bar graph to compare

Applications :

1. Customer segmentation : Agglomerative or divisive clustering can be used to group customers into different clusters based on their demographic, spending, and other characteristics. This can be used to better understand customer behavior and to target marketing campaigns.

2. Image segmentation : Hierarchical clustering can be used to segment images into different regions, which can then be used for further analysis.

3. Text analysis : Hierarchical clustering can be used to group text documents based on their content, which can then be used for text mining or text classification tasks.

4. Gene expression analysis : Hierarchical clustering can be used to group genes based on their expression levels, which can then be used to better understand gene expression patterns.

5. Anomaly detection : Hierarchical clustering can be used to detect anomalies in data, which can then be used for fraud detection or other tasks.

6. Recommendation systems : Hierarchical clustering can be used to group users based on their preferences, which can then be used to recommend items to them.

7. Risk assessment : Agglomerative or divisive clustering can be used to group different risk factors in order to better understand the overall risk of a portfolio.

8. Network analysis : Hierarchical clustering can be used to group nodes in a network based on their connections, which can then be used to better understand network structures.

9. Market segmentation : Hierarchical clustering can be used to group markets into different segments, which can then be used to target different products or services to them.

10. Outlier detection : Hierarchical clustering can be used to detect outliers in data, which can then be used for further analysis.

Conclusion :

Thus we have implemented Agglomerative hierarchical clustering algorithm using appropriate dataset

Outcome :

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform Agglomerative hierarchical clustering algorithm.

Questions :

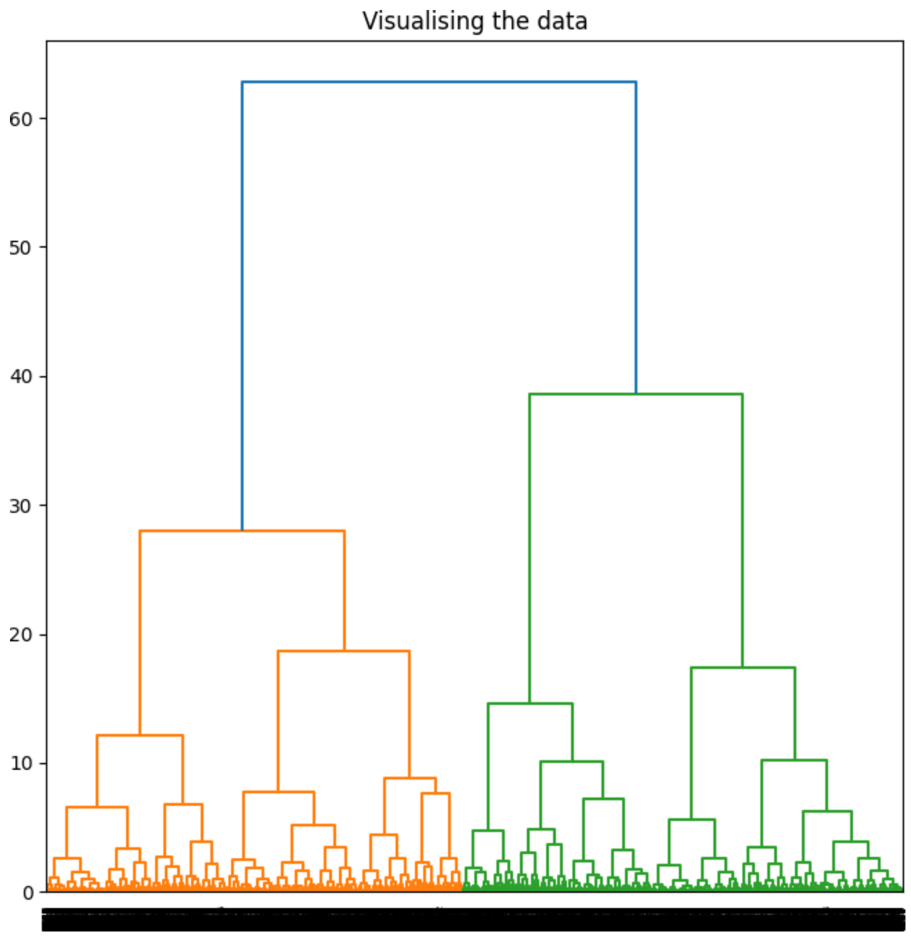
1. What are the two types of hierarchical clustering ?
2. What is divisive clustering ?
3. State minimum five advantages of hierarchical clustering.
4. What are the disadvantages of hierarchical clustering ?
5. What are the common algorithms used for clustering ?
6. What is a dendogram ?

Name : Nikhil Sonone **Roll No :** 68

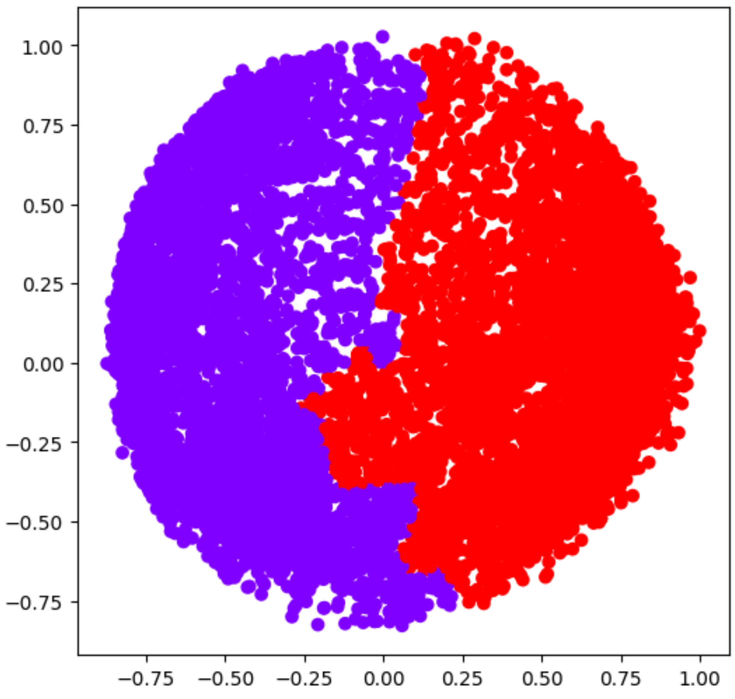
Academic Year – 2023 – 24 **Class and Department –** BE AI&DS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc

X = pd.read_csv('./datasets/CC_GENERAL.csv')

# Dropping the CUST_ID column from the data
X = X.drop('CUST_ID', axis = 1)

# Handling the missing values
X.fillna(method ='ffill', inplace = True)

# Scaling the data so that all the features become comparable
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalizing the data so that the data approximately follows a
Gaussian distribution
X_normalized = normalize(X_scaled)

# Converting the numpy array into a pandas
DataFrameX_normalized = pd.DataFrame(X_normalized)

# Reducing the dimensionality of the Data
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
```

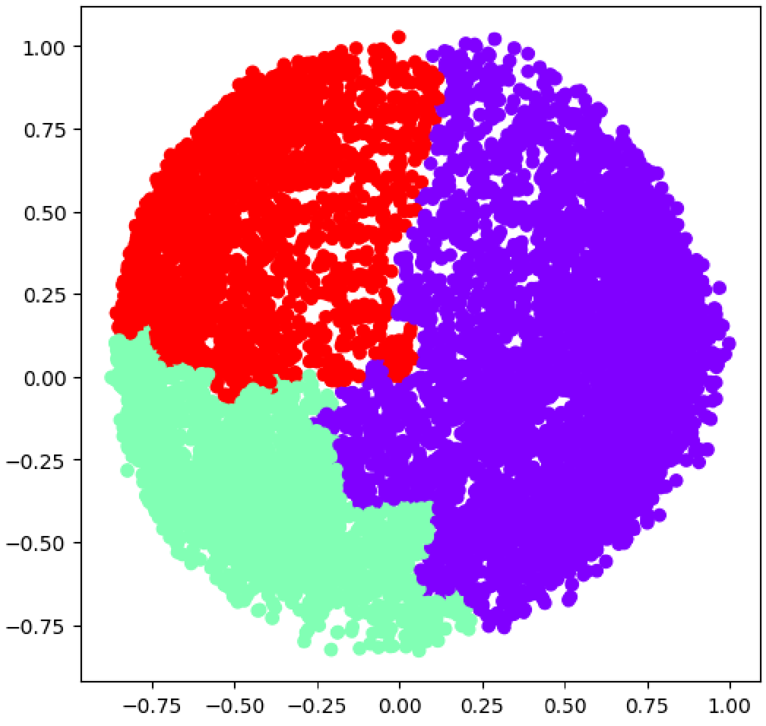
```
# Visualizing theworking of the Dendrograms
# Dendrograms are used to divide a given clusterinto many different
clusters
plt.figure(figsize =(8, 8))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram((shc.linkage(X_principal, method
='ward')))
```



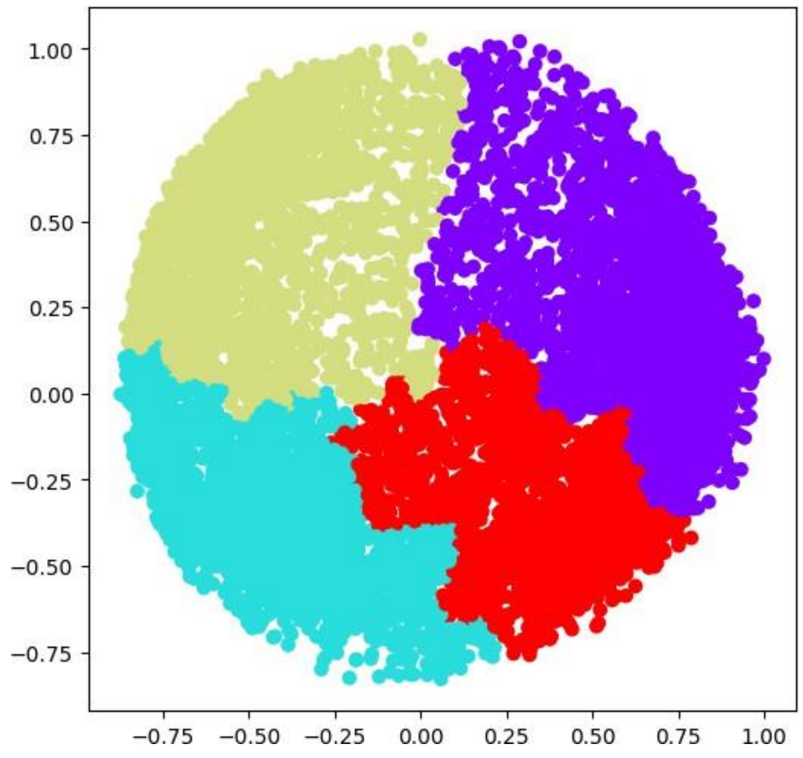
```
# Building and Visualizing the different clustering models for
different values of k
# k = 2
ac2 = AgglomerativeClustering(n_clusters = 2)
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
c = ac2.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



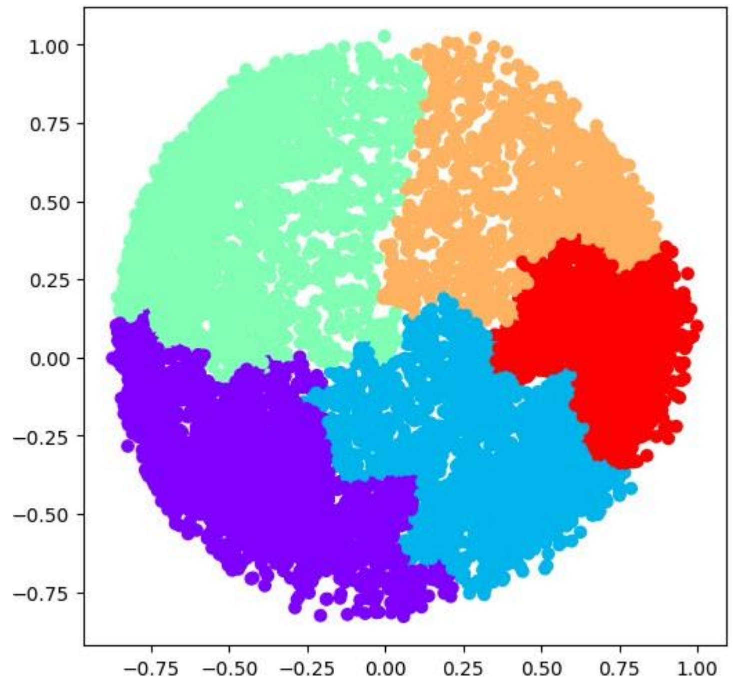
```
# k = 3
ac3 = AgglomerativeClustering(n_clusters = 3)
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'],
X_principal['P2'],
c = ac3.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



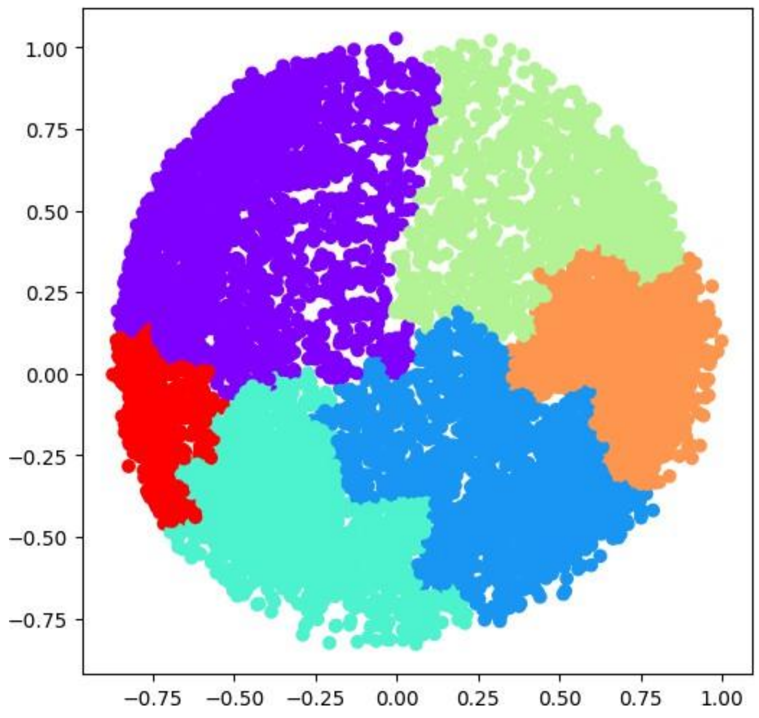
```
# k = 4
ac4 = AgglomerativeClustering(n_clusters = 4)
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'],
X_principal['P2'],
c = ac4.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



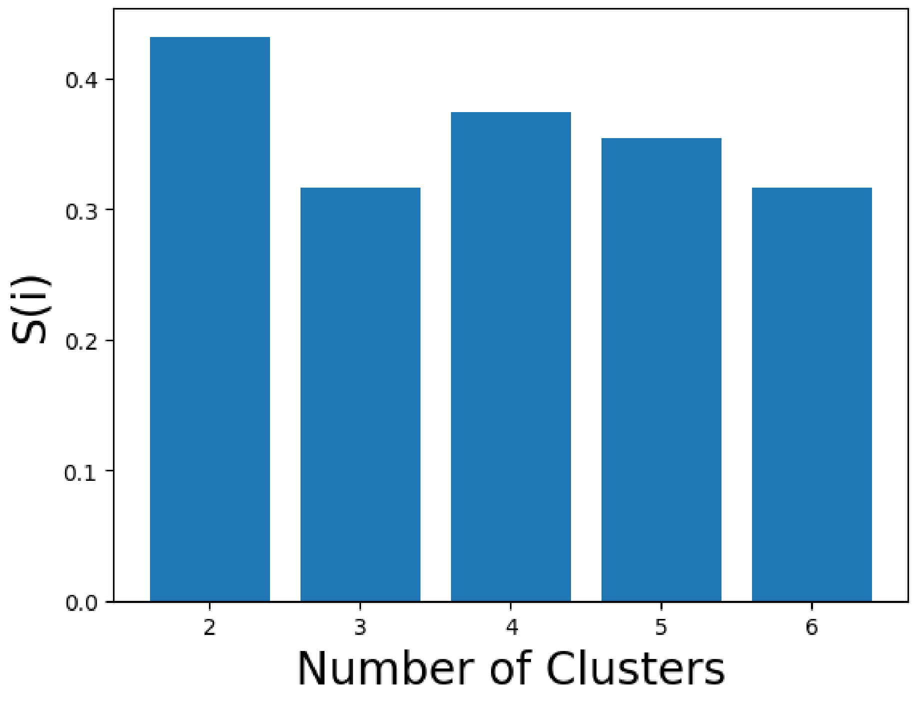
```
# k = 5
ac5 = AgglomerativeClustering(n_clusters = 5)
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'],
X_principal['P2'],
c = ac5.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



```
# k = 6
ac6 = AgglomerativeClustering(n_clusters = 6)
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'],
X_principal['P2'],
c = ac6.fit_predict(X_principal), cmap ='rainbow')
plt.show()
```



```
# Evaluating the different models and Visualizing the results.
k = [2, 3, 4, 5, 6]

# Appending the silhouette scores of the different models to the
list
silhouette_scores = []
silhouette_scores.append(silhouette_score(X_principal,
ac2.fit_predict(X_principal)))
silhouette_scores.append(silhouette_score(X_principal,
ac3.fit_predict(X_principal)))
silhouette_scores.append(silhouette_score(X_principal,
ac4.fit_predict(X_principal)))
silhouette_scores.append(silhouette_score(X_principal,
ac5.fit_predict(X_principal)))
silhouette_scores.append(silhouette_score(X_principal,
ac6.fit_predict(X_principal)))

# Plotting a bar graph to compare the results
plt.bar(k,silhouette_scores)
plt.xlabel('Number of Clusters', fontsize = 20)
plt.ylabel('S(i)',fontsize = 20)
plt.show()
```



Practical No. 05

Aim :

Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Objective :

To Study :

1. Implementation of Page Rank Algorithm using Python or Beautiful Soup

Theory :

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known. The above centrality measure is not implemented for multi-graphs.

Algorithm :

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called “iterations”, through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Simplified Algorithm :

Assume a small universe of four web pages: A, B, C, and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25.

The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.

If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value, 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one-third of its existing value, or approximately 0.083, to A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.

In other words, the PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links L( ).

In the general case, the PageRank value for any page u can be expressed as :

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v. The algorithm involves a damping factor for the calculation of the PageRank. It is like the income tax which the govt extracts from one despite paying him itself.

Applications :

Used for ranking of location in Planning and control of production and logistics in large companies.

Conclusion :

Thus we have implemented page ranking algorithm.

Outcome :

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): implement page ranking algorithm and measure the importance of website pages.

Questions :

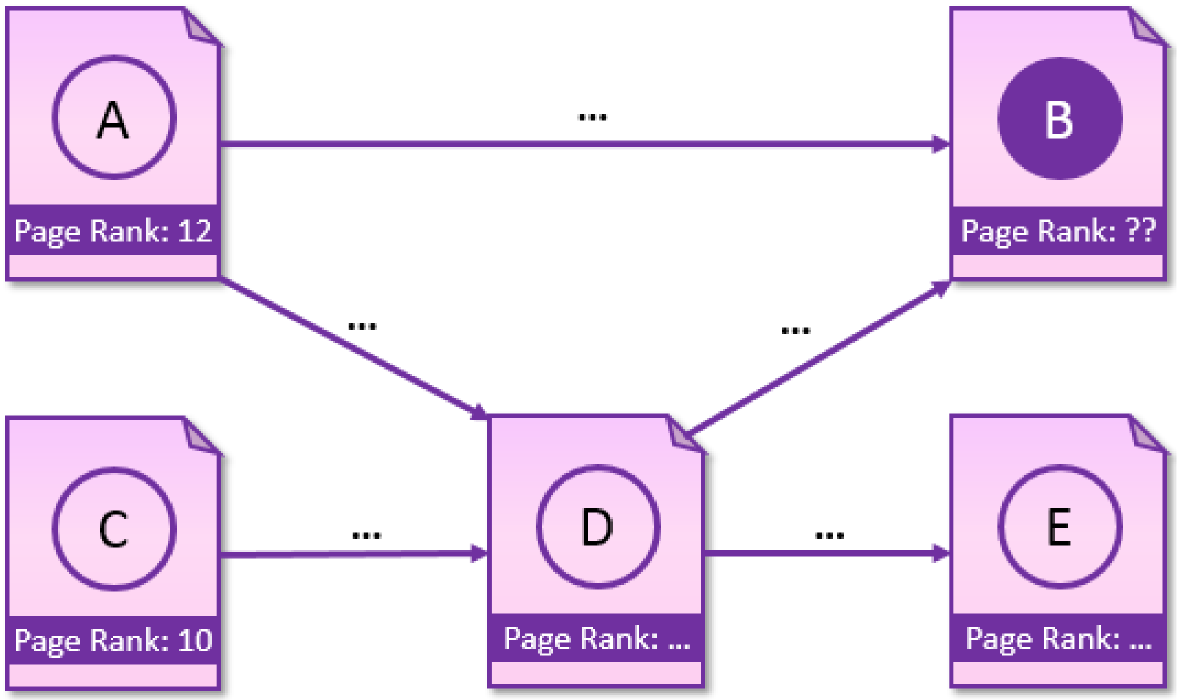
1. What is page ranking algorithm?

2. What was Page ranking algorithm named after?

3. How to improve PageRank Algorithm?

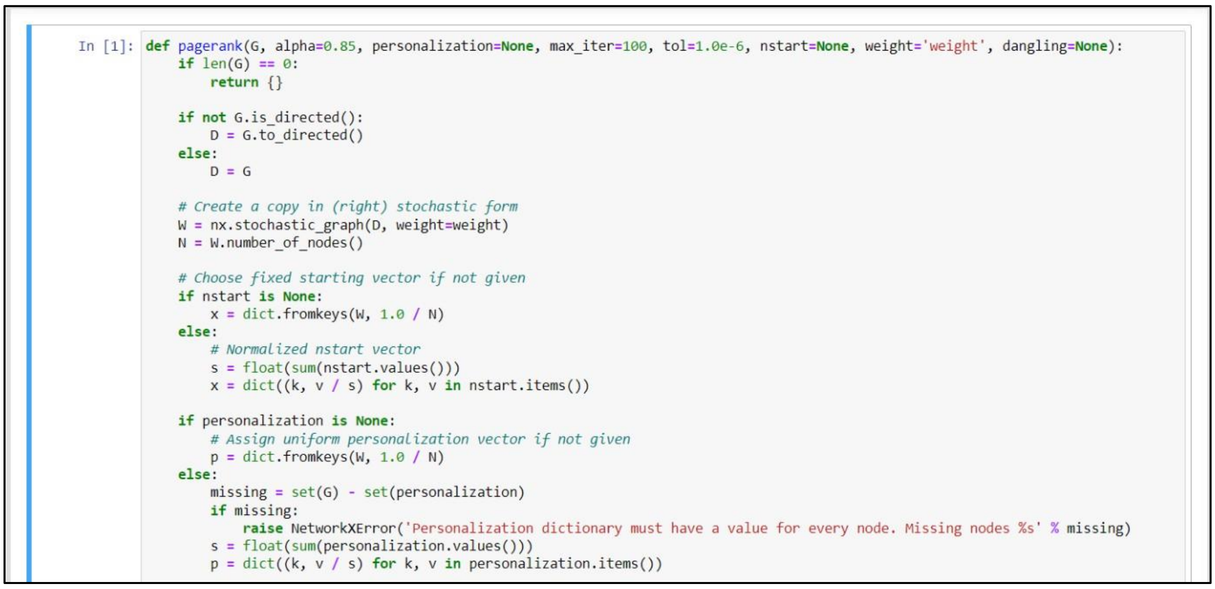
4. How Google determines a page’s relevance or importance.

5. Find the page rank score of web page B:

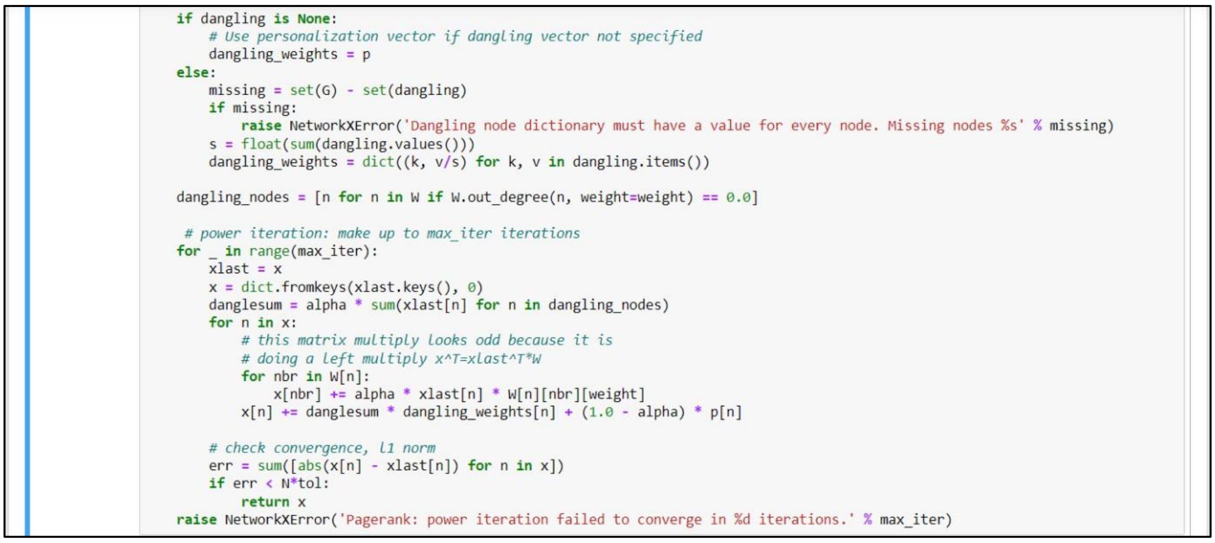


Name : Nikhil Sonone **Roll No :** 68

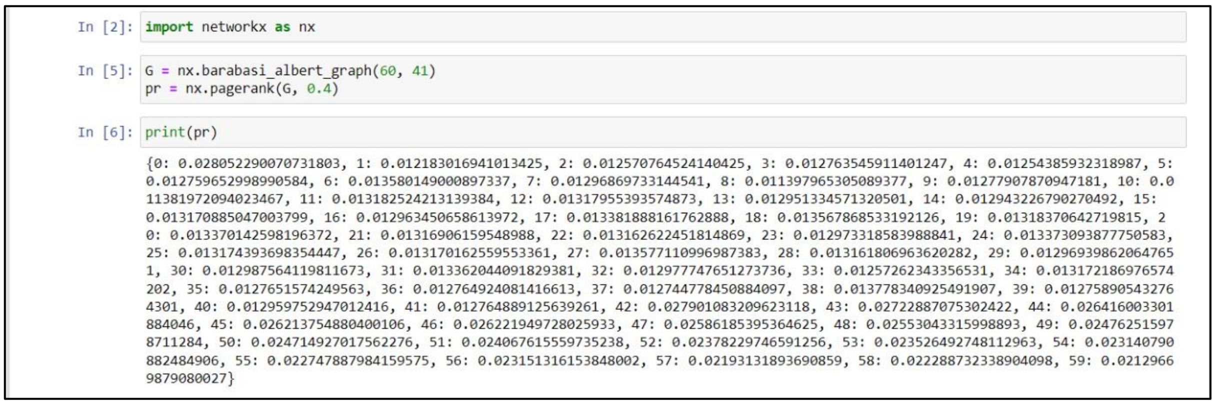
Academic Year – 2023 – 24 **Class and Department –** BE AI&DS



```
In [1]: def pagerank(G, alpha=0.85, personalization=None, max_iter=100, tol=1.0e-6, nstart=None, weight='weight', dangling=None):
            if len(G) == 0:
                return {}

            if not G.is_directed():
                D = G.to_directed()
            else:
                D = G

            # Create a copy in (right) stochastic form
            W = nx.stochastic_graph(D, weight=weight)
            N = W.number_of_nodes()

            # Choose fixed starting vector if not given
            if nstart is None:
                x = dict.fromkeys(W, 1.0 / N)
            else:
                # Normalized nstart vector
                s = float(sum(nstart.values()))
                x = dict((k, v / s) for k, v in nstart.items())

            if personalization is None:
                # Assign uniform personalization vector if not given
                p = dict.fromkeys(W, 1.0 / N)
            else:
                missing = set(G) - set(personalization)
                if missing:
                    raise NetworkXError('Personalization dictionary must have a value for every node. Missing nodes %s' % missing)
                s = float(sum(personalization.values()))
                p = dict((k, v / s) for k, v in personalization.items())
```



```
            if dangling is None:
                # Use personalization vector if dangling vector not specified
                dangling_weights = p
            else:
                missing = set(G) - set(dangling)
                if missing:
                    raise NetworkXError('Dangling node dictionary must have a value for every node. Missing nodes %s' % missing)
                s = float(sum(dangling.values()))
                dangling_weights = dict((k, v/s) for k, v in dangling.items())

            dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

             # power iteration: make up to max_iter iterations
            for _ in range(max_iter):
                xlast = x
                x = dict.fromkeys(xlast.keys(), 0)
                danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
                for n in x:
                    # this matrix multiply looks odd because it is
                    # doing a left multiply x^T=xlast^T*W
                    for nbr in W[n]:
                        x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
                    x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]

                # check convergence, l1 norm
                err = sum([abs(x[n] - xlast[n]) for n in x])
                if err < N*tol:
                    return x
            raise NetworkXError('Pagerank: power iteration failed to converge in %d iterations.' % max_iter)
```



```
In [2]: import networkx as nx

In [5]: G = nx.barabasi_albert_graph(60, 41)
        pr = nx.pagerank(G, 0.4)

In [6]: print(pr)

        {0: 0.028052290070731803, 1: 0.012183016941013425, 2: 0.012570764524140425, 3: 0.012763545911401247, 4: 0.01254385932318987, 5:
        0.012759652998990584, 6: 0.013580149000897337, 7: 0.01296869733144541, 8: 0.011397965305089377, 9: 0.01277907870947181, 10: 0.0
        11381972094023467, 11: 0.013182524213139384, 12: 0.01317955393574873, 13: 0.012951334571320501, 14: 0.012943226790270492, 15:
        0.013170885047003799, 16: 0.012963450658613972, 17: 0.013381888161762888, 18: 0.013567868533192126, 19: 0.01318370642719815, 2
        0: 0.013370142598196372, 21: 0.01316906159548988, 22: 0.013162622451814869, 23: 0.012973318583988841, 24: 0.013373093877750583,
        25: 0.013174393698354447, 26: 0.013170162559553361, 27: 0.013577110996987383, 28: 0.013161806963620282, 29: 0.01296939862064765
        1, 30: 0.012987564119811673, 31: 0.013362044091829381, 32: 0.012977747651273736, 33: 0.01257262343356531, 34: 0.013172186976574
        202, 35: 0.0127651574249563, 36: 0.012764924081416613, 37: 0.012744778450884097, 38: 0.013778340925491907, 39: 0.01275890543276
        4301, 40: 0.012959752947012416, 41: 0.012764889125639261, 42: 0.027901083209623118, 43: 0.02722887075302422, 44: 0.026416003301
        884046, 45: 0.026213754880400106, 46: 0.026221949728025933, 47: 0.02586185395364625, 48: 0.02553043315998893, 49: 0.0247625159
        8711284, 50: 0.024714927017562276, 51: 0.024067615559735238, 52: 0.02378229746591256, 53: 0.023526492748112963, 54: 0.023140790
        882484906, 55: 0.022747887984159575, 56: 0.023151316153848002, 57: 0.02193131893690859, 58: 0.022288732338904098, 59: 0.0212966
        9879080027}
```