TEXT AND WEB INTELLIGENCE ANALYTICS

**LAB MANUAL**

|  |  |
| --- | --- |
| **SUBMITTED BY** | **MUSKAN NANDKANI** |
| **ROLL NUMBER** | **17CSU121** |
| **CLASS** | **CSE-VIII** |
| **SESSION** | **2020-2021** |
| **FACULTY** | **DR. VAISHALI KALRA** |
|  |  |



# Department of Computer Science and Engineering The Northcap University, Sector-23, Gurugram, Haryana

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **COURSE-CURRICULUM EXPERIMENTS** | | | | |
| **S.NO.** | **Experiment** | **Date** | **Grade** | **Signature** |
| 1 | Frequency Distribution | 29-01-2021 |  |  |
| 2 | Stopwords | 05/02/2021 |  |  |
| 3 | Exploring Names Corpus | 12/02/2021 |  |  |
| 4 | Similarity metrices | 16/02/2021 |  |  |
| 5 | Lesk Algorithm for Word Sense Disambiguation | 24/02/2021 |  |  |
| 6 | Implement LDA with BOW and TF-IDF features and  compare the results | 01/04/2021 |  |  |
| 7 | Implementation of KNN, Naive  Bayes and Multinominal Naive Bayes. | 08/04/2021 |  |  |
| 8 | Implementation of K means, K Medoids and Hierarchical Clustering algorithms. | 02/05/2021 |  |  |
| 9 | Implement homophily for social media analysis. | 09/05/2021 |  |  |
| **VALUE ADDED EXPERIMENTS** | | | | |
| 1 | Chunking, Chinking and Named Entity Recognition | 03/03/2021 |  |  |
| 2 | Tokenization and Exploring Gutenberg Corpus | 22-01-2021 |  |  |

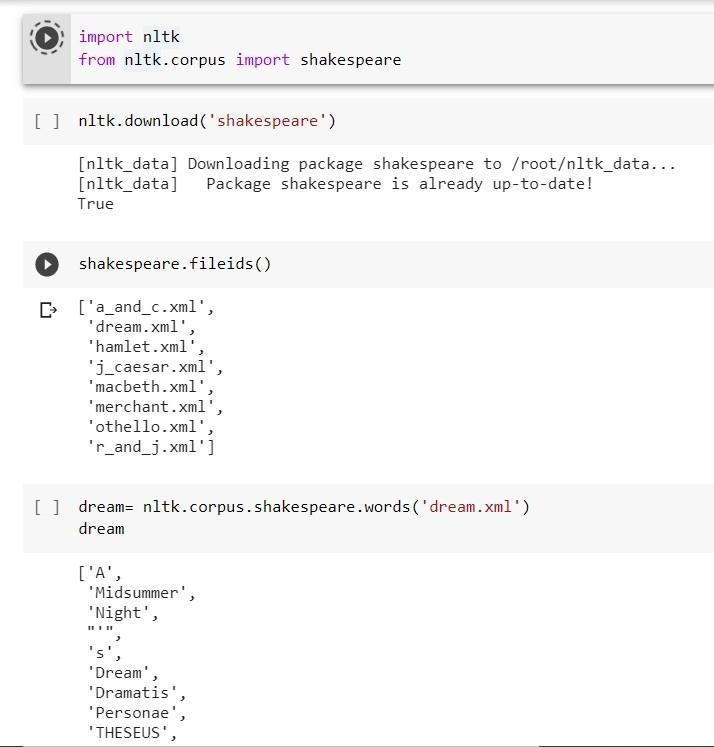
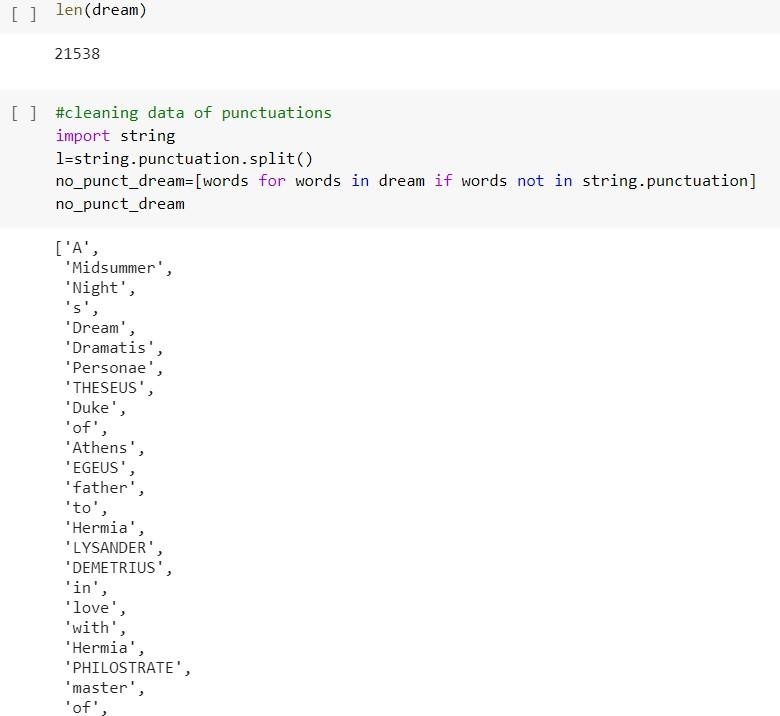
# EXPERIMENT NO. 1

**TWIA Lab Manual (CSL 554)**

# 2020-21

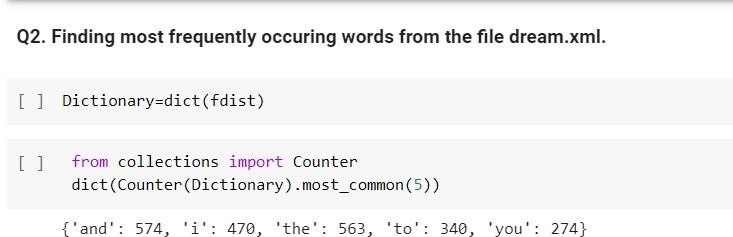
|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 29 January 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objectives:**   1. Import the corpus Shakespeare and find the frequency of each word in the file dream.xml. 2. Find 5 most frequently occurring words from the file dream.xml. 3. Import wordnet corpus from the available nltk corpus list and find out the sysnset of word bank. Also find the definition and example of first sysnset in the list.   **Background Study:**  **nltk.FreqDist( )**  A frequency distribution records the number of times each outcome of an experiment has occurred. For example, a frequency distribution could be used to record the frequency of each word type in a document. Formally, a frequency distribution can be defined as a function mapping from each sample to the number of times that sample occurred as an outcome.  **Wordnet**  Wordnet is a lexical database of semantic relations between words in more than 200 languages. WordNet links words into semantic relations including synonyms, hyponyms, and meronym. |
| **Outcome:** Students will be able to learn the concepts of nltk.freqdist(), collections in python and sysnets in wordnet library. |
| **Problem Statement:**  **1. Import the corpus Shakespeare and find the frequency of each word in the file dream.xml.**  **CODE AND OUTPUT:** |



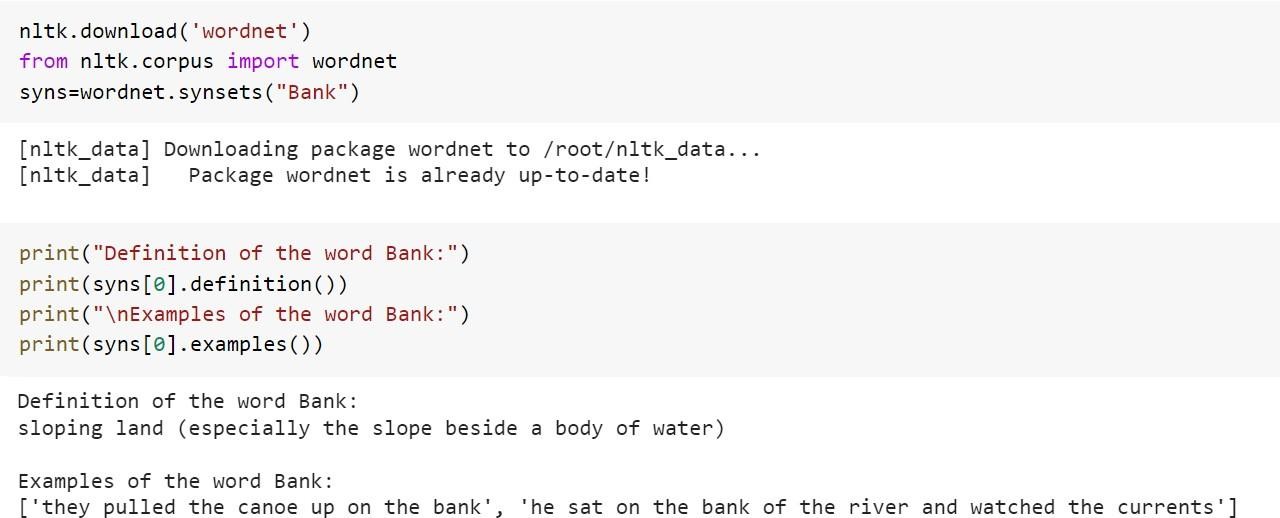


1. **Find 5 most frequently occurring words from the file dream.xml. CODE AND OUTPUT:**



# Import wordnet corpus from the available nltk corpus list and find out the sysnset of word bank. Also find the definition and example of first sysnset in the list.

**CODE AND OUTPUT:**



# EXPERIMENT NO. 2

**TWIA Lab Manual (CSL 554)**

# 2020-21

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 5 February 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:**   1. Print all the Arabic Stopwords. 2. Omit a given list of stop words from the total stopwords list of English language. |
| **Background Study:**  **Stopwords**  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 are considered stopping words.  NLTK (Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. |
| **Outcome:** Students will be able to understand the concept of stopwords in nltk and list comprehensions in python.  **Problem Statement:**  **1. Print all the Arabic Stopwords. CODE AND OUTPUT:** |

1. **Omit a given list of stop words from the total stopwords list of English language.**

# CODE AND OUTPUT:



**TWIA Lab Manual (CSL 554)**

# 2020-21

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 12 February 2021** |
| **Faculty Signature:** |
| **Grade:** |

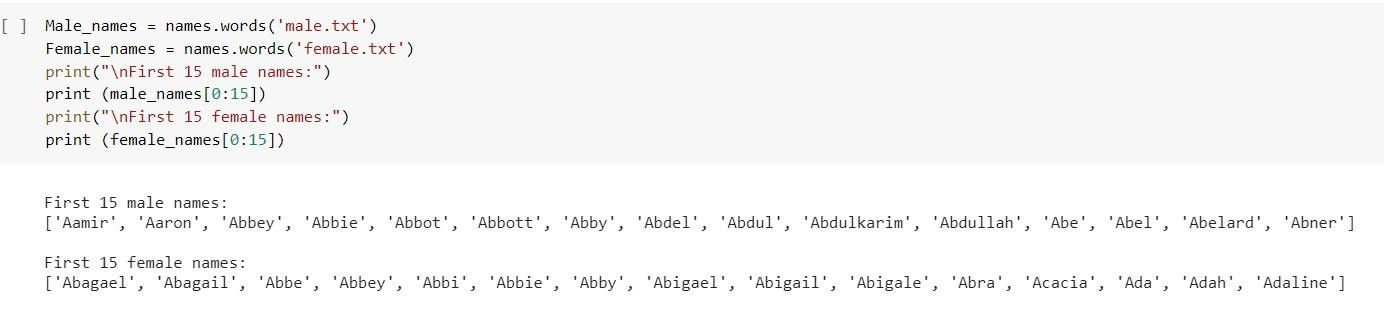
|  |
| --- |
| **Objectives:**   1. Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names. 2. From the names corpus, combine all the labelled male and female names and print any 20. 3. Print the definition and examples of any one English language word using WordNet corpus.   **Background Study:**  **Text Corpus**  A text corpus is a large and structured set of texts (nowadays usually electronically stored and processed). Text corpora are used to do statistical analysis and hypothesis testing, checking occurrences, or validating linguistic rules within a specific language territory.  **Wordnet Corpus**  Wordnet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (sysnets), each expressing a distinct concept. Sysnets are interlinked by means of conceptual-semantic and lexical relations.  WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. |
| **Outcome:** Students will be able to explore names corpus, understand the concept of labelling the data and learn about sysnets in Wordnet corpus. |
| **Problem Statement:** |

# Print the total number of male and female names in the names corpus. Then, Print the first 15 male and female names.

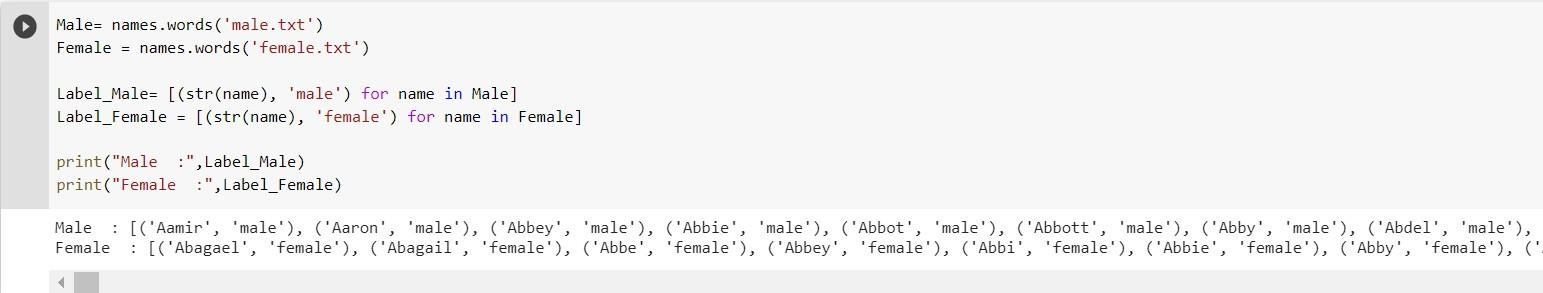
**CODE AND OUTPUT:**

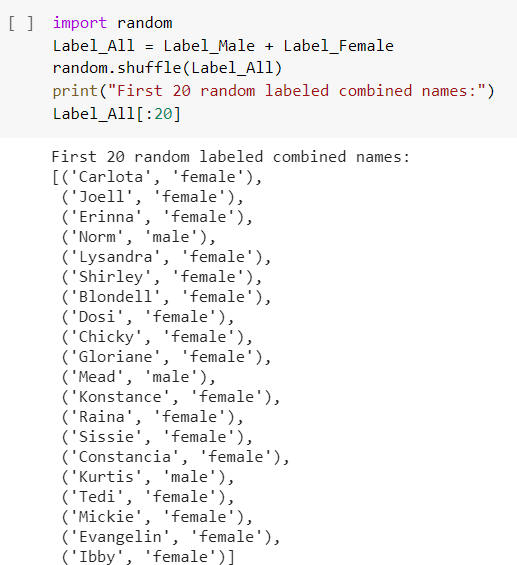


# From the names corpus, combine all the labelled male and female names and print any 20.

**CODE AND OUTPUT:**

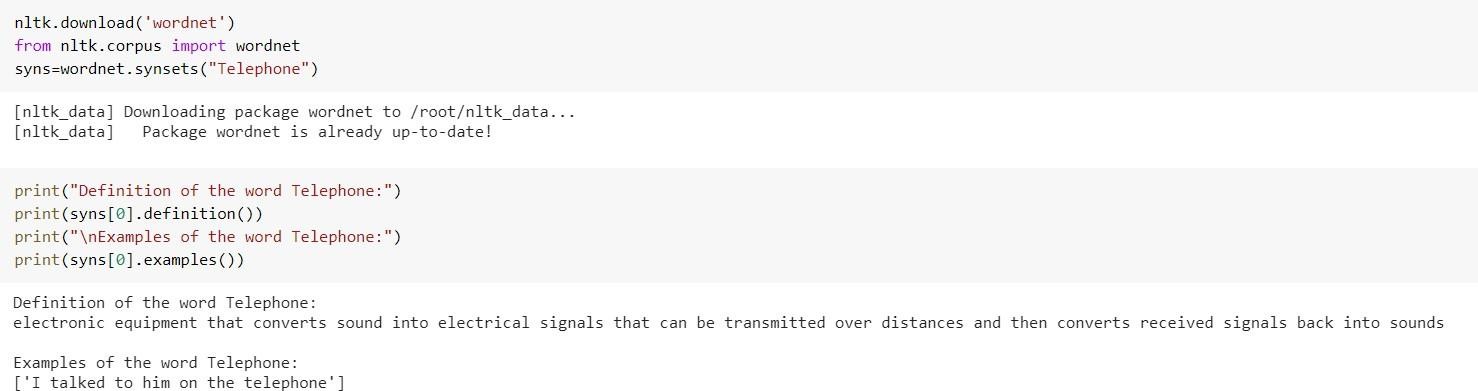
**CODE AND OUTPUT:**





# Print the definition and examples of any one English language word using WordNet corpus.

**CODE AND OUTPUT:**



# TWIA Lab Manual (CSL 554)

**2020-21**

# EXPERIMENT NO. 4

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:**  [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 16 February 2021** |
| **Faculty Signature:** |
| **Grade:** |

**Objective:** To implement Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer.

# Background Study:

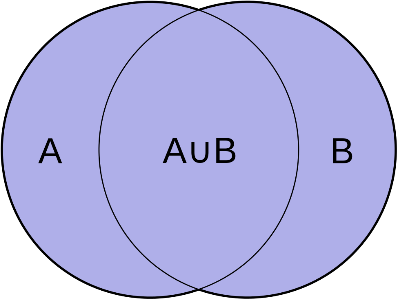
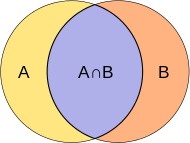
**Levenshtein distance**

The Levenshtein distance is a string metric for measuring difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

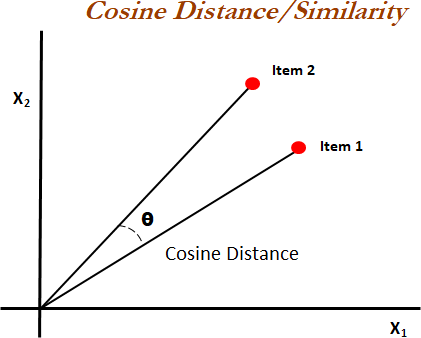
# Jaccard similarity

The Jaccard index, also known as the Jaccard similarity coefficient, is a [statistic](https://en.wikipedia.org/wiki/Statistic) used for gauging

the [similarity](https://en.wikipedia.org/wiki/Similarity_measure) and [diversity](https://en.wikipedia.org/wiki/Diversity_index) of [sample](https://en.wikipedia.org/wiki/Sample_(statistics)) sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the [intersection](https://en.wikipedia.org/wiki/Intersection_(set_theory)) divided by the size of the [union](https://en.wikipedia.org/wiki/Union_(set_theory)) of the sample sets.



# Cosine similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

# TF-IDF Vectorizer

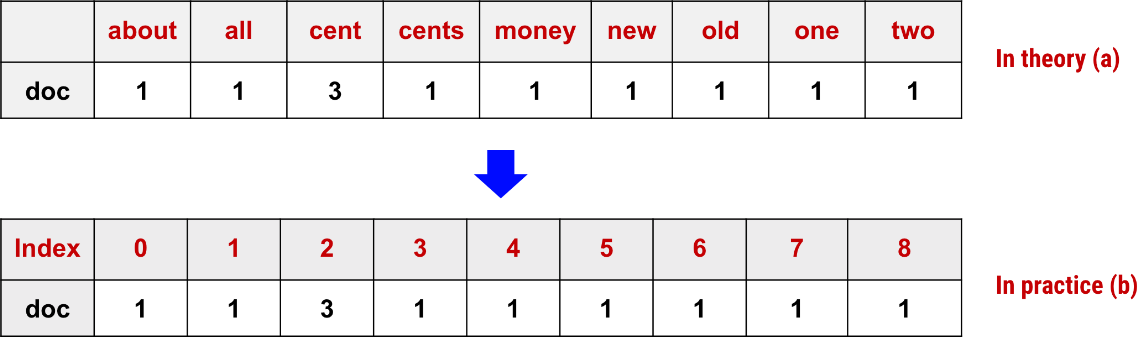
TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

TF-IDF for a word in a document is calculated by multiplying two different metrics:

* The **term frequency** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.
* The **inverse document frequency** of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.
* So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

# CountVectorizer

**CountVectorizer** is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

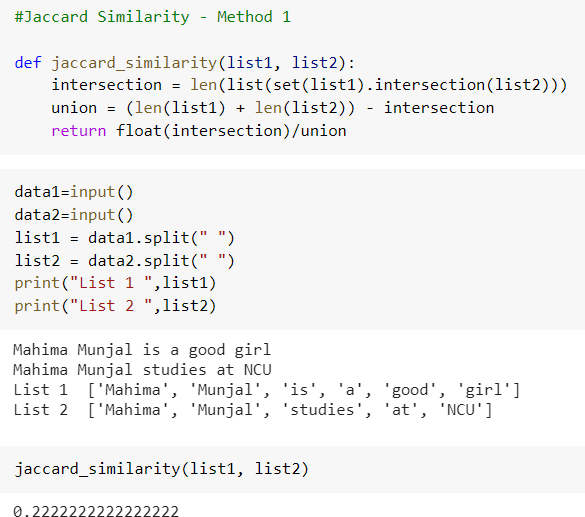


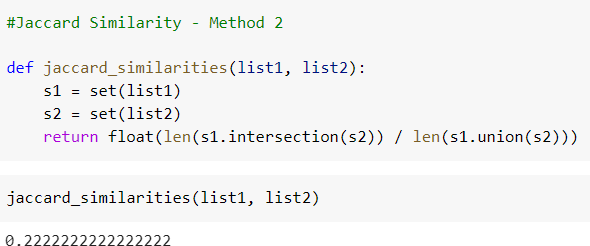
**Outcome:** Students will be able to demonstrate Levenshtein Edit Distance , Jaccard similarity , Cosine Similarity using both TF-IDF and count vectorizer.

# Problem Statement:

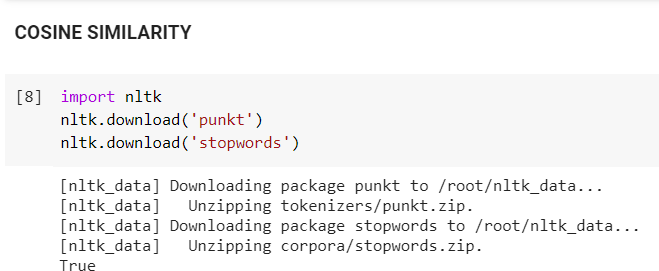
1. **Demonstrate the computation of Similarity Metrics such as Jaccard, Levenshtein and Cosine.**

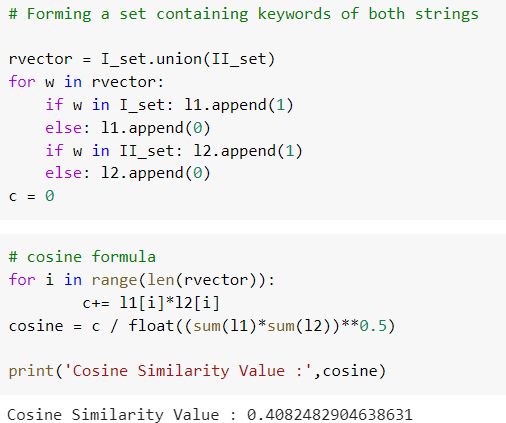
# CODE AND OUTPUT :



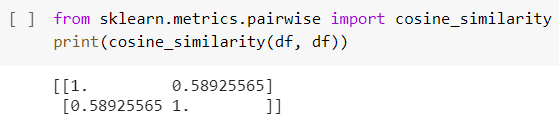








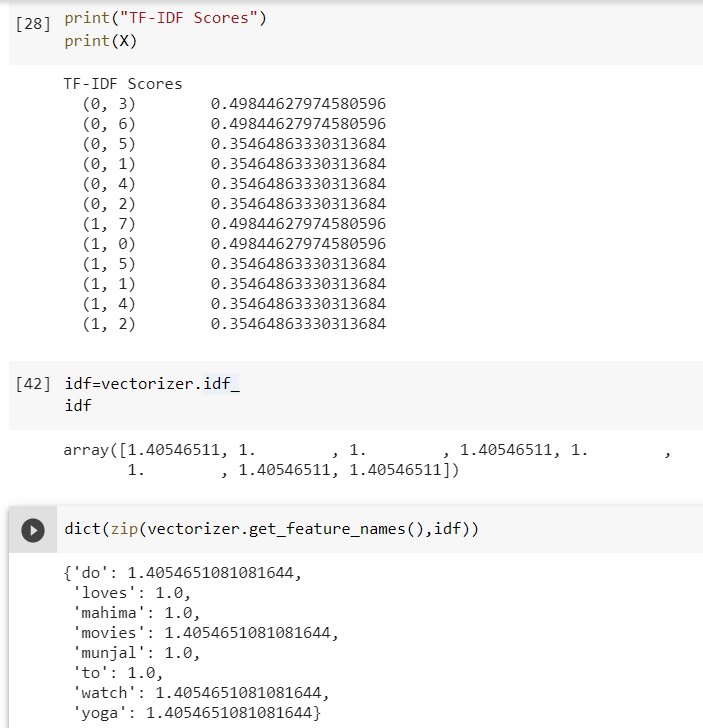




1. **Calculate the TF-IDF vectorizer on 2 documents.**

# CODE AND OUTPUT:





1. **Apply the max-df, min-df param in the TF-IDF function.**

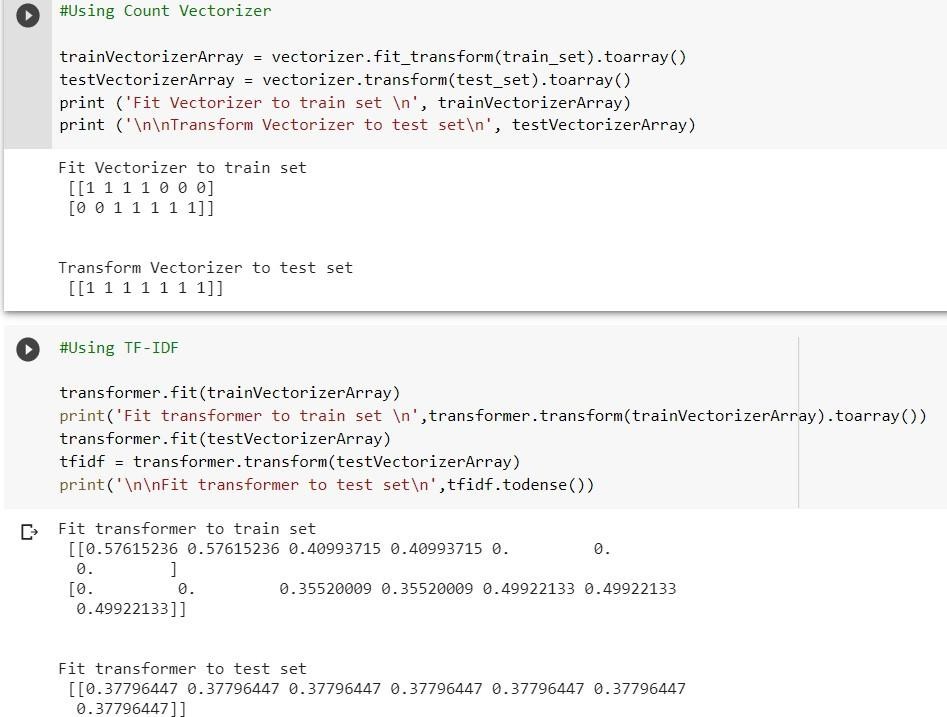
# CODE AND OUTPUT:



1. **Compute Cosine Similarity using both TF-IDF and Count vectorizer CODE AND OUTPUT:**







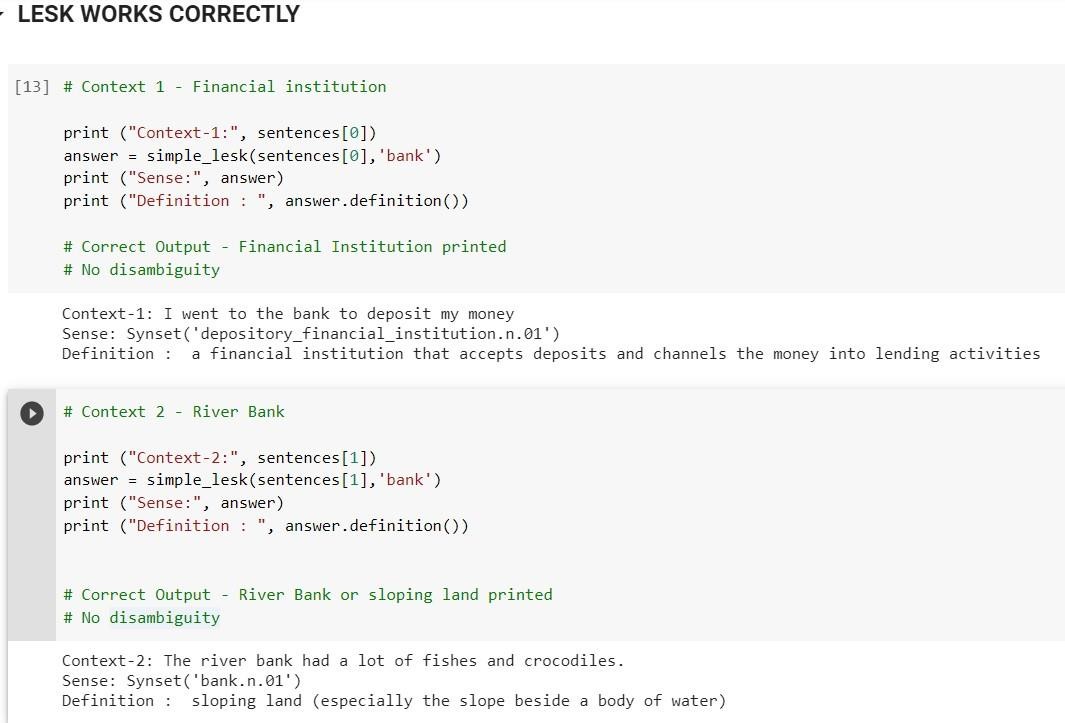
# EXPERIMENT NO. 5

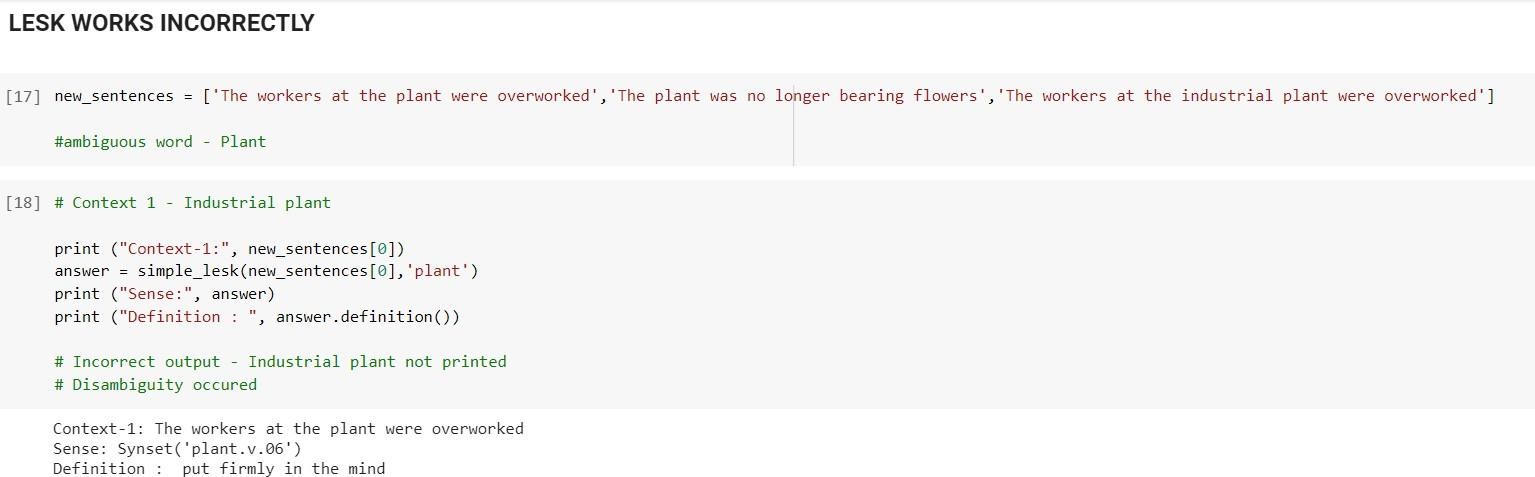
**TWIA Lab Manual (CSL 554)**

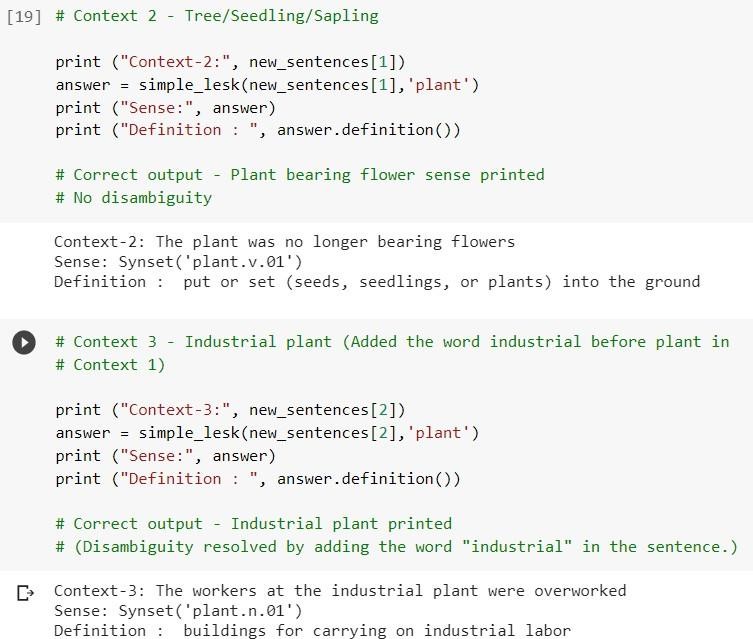
# 2020-21

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 24 February 2021** |
| **Faculty Signature:** |
| **Grade:** |

|  |
| --- |
| **Objective:** Implementation of the Lesk algorithm for Word Sense Disambiguation.  **Background Study:**  **Lesk Algorithm**  The Lesk algorithm assumes that words in each "neighbourhood" (section of text) will tend to share a common topic. A simplified version of the Lesk algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighbourhood.  An implementation might look like this:   1. for every sense of the word being disambiguated one should count the number of words that are in both neighbourhood of that word and in the dictionary definition of that sense 2. the sense that is to be chosen is the sense which has the biggest number of this count. |
| **Outcome:** Students will be able to demonstrate how to Lesk algorithm works. **Problem Statement:** Implement Lesk algorithm for Word Sense Disambiguation. **CODE AND OUTPUT:** |









**EXPERIMENT NO. 6**

# TWIA Lab Manual (CSL 554)

**2020-21**

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 01 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

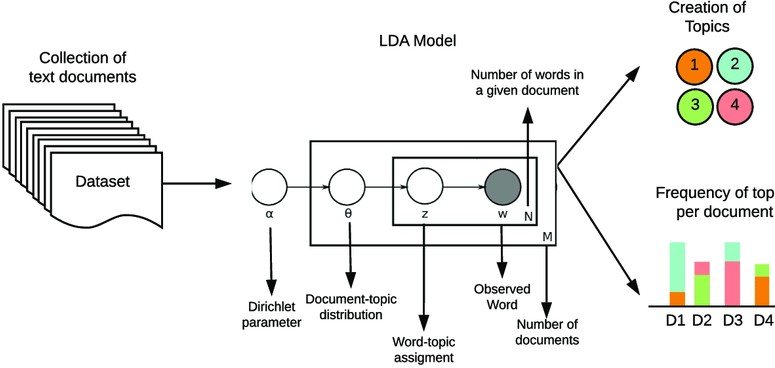
# Objectives:

* 1. Implement LDA with Bag of Words
  2. Implement LDA with TF-IDF
  3. Compare the accuracy using Score values make the analysis. **Dataset:** [**https://www.kaggle.com/therohk/million-headlines**](https://www.kaggle.com/therohk/million-headlines) **Background Study:**

# [Latent Dirichlet Allocation](http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/) (LDA)

It is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.

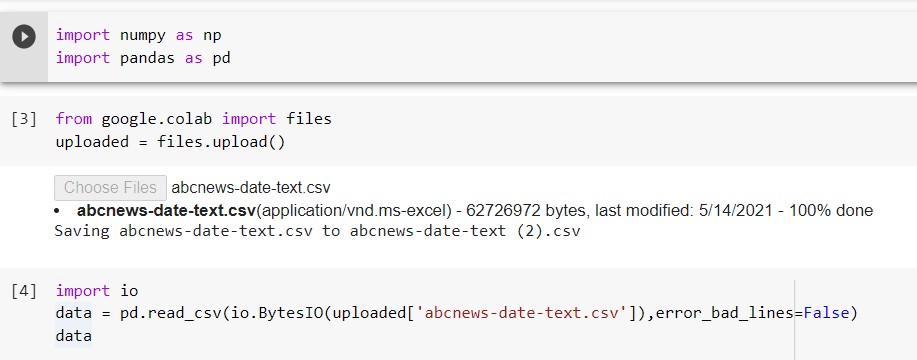
For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a [topic model](https://en.wikipedia.org/wiki/Topic_model) and belongs to the [machine learning](https://en.wikipedia.org/wiki/Machine_learning) toolbox and in wider sense to the [artificial](https://en.wikipedia.org/wiki/Artificial_intelligence) [intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) toolbox.

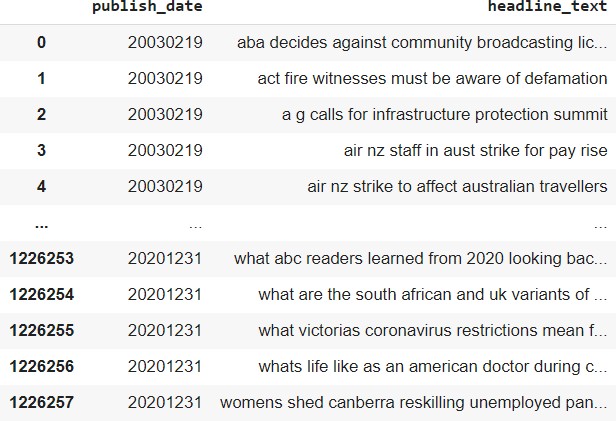


**Outcome:** Students will be able to demonstrate how LDA works using Bag of Words and using TF-IDF.

**Problem Statement:** Implement LDA using BOW and TF-IDF on Million News Dataset

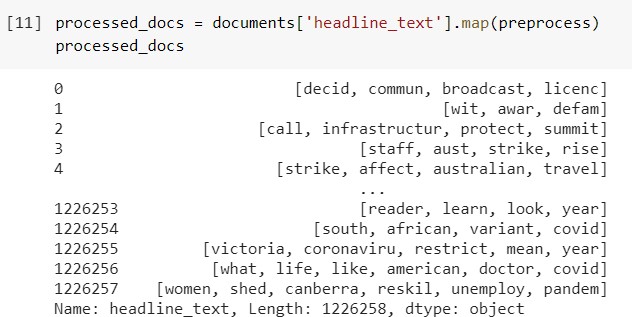
# CODE AND OUTPUT:

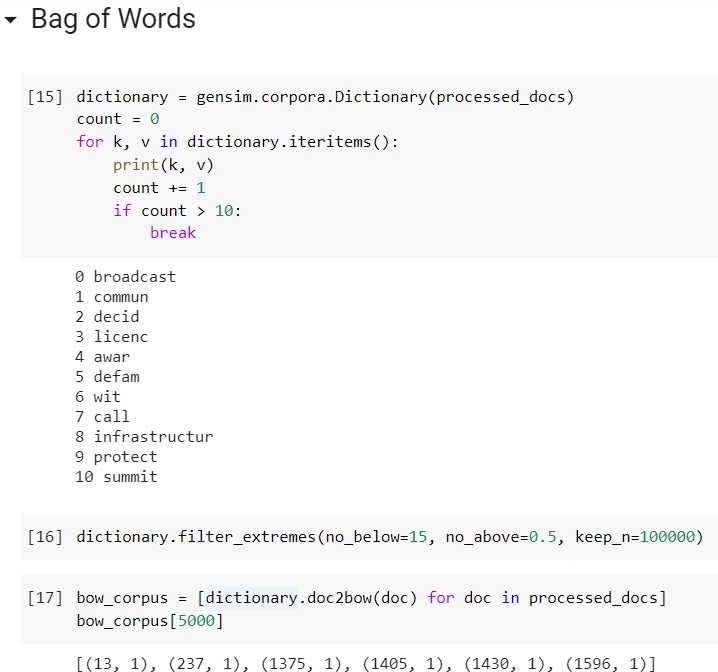


















**EXPERIMENT NO. 7**

# TWIA Lab Manual (CSL 554)

**2020-21**

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 4 April 2021** |
| **Faculty Signature:** |
| **Grade:** |

# Objectives:

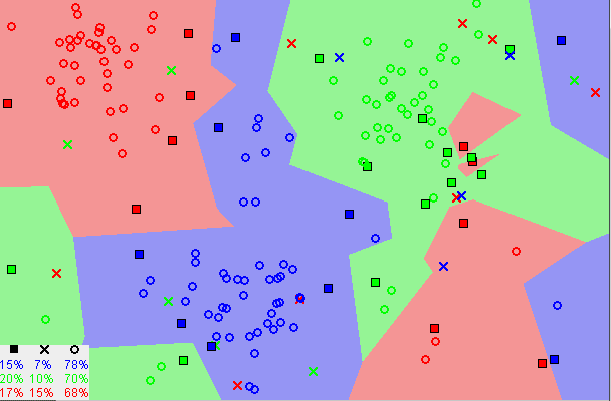
1. **Perform KNN, Naïve Bayes and Multinomial Naïve Bayes on 20 Newsgroup Dataset.**

**Dataset:** [**https://www.kaggle.com/crawford/20-newsgroups**](https://www.kaggle.com/crawford/20-newsgroups)

**Background Study:**

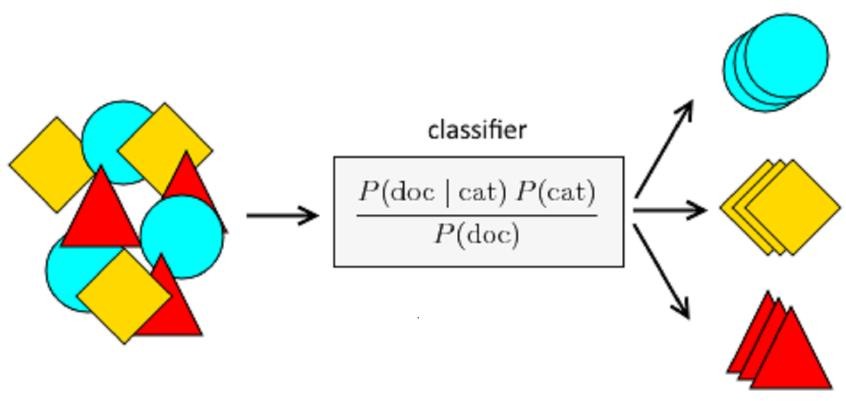
# K- Nearest Neighbours(KNN)

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

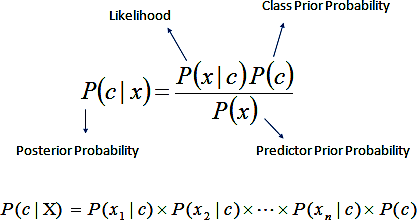


# Naïve Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.



# Multinomial Naïve Bayes

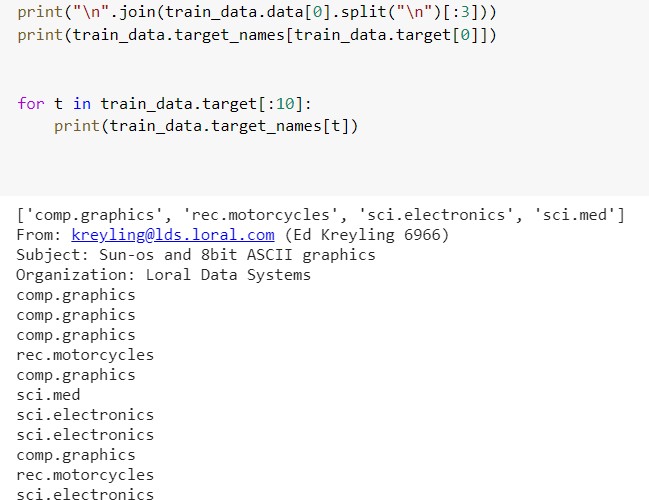
The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

**Outcome:** Students will be able to understand the difference between the implementations of KNN, Naïve Bayes and Multinomial Naïve Bayes.

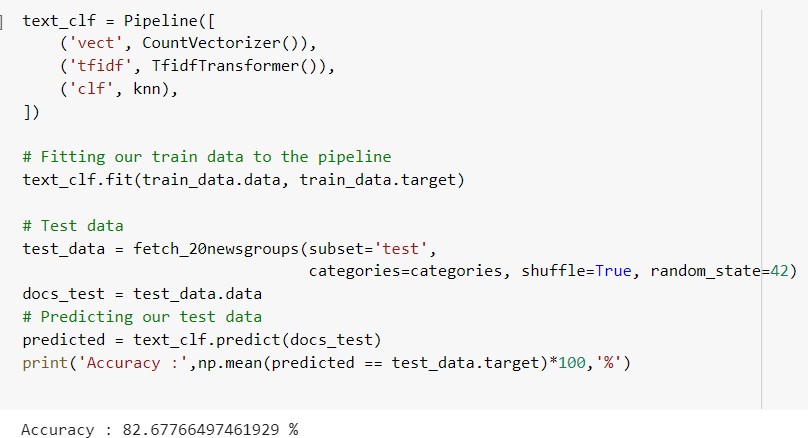
**Problem Statement:** Implement KNN, Naïve Bayes and Multinomial Naïve Bayes on20 Newsgroup datset and compare their accuracy scores.

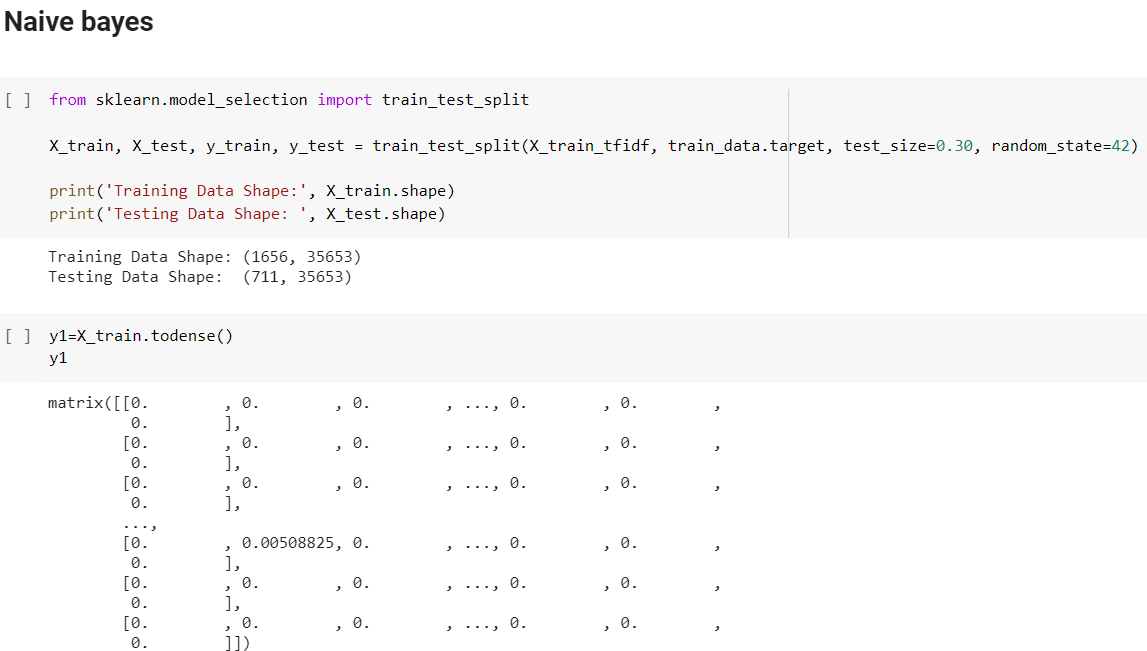
# CODE AND OUTPUT:

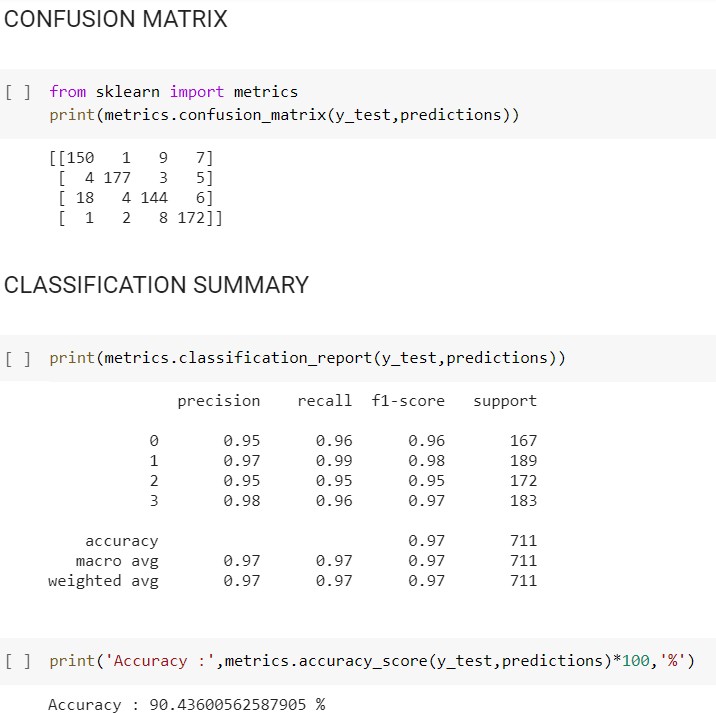
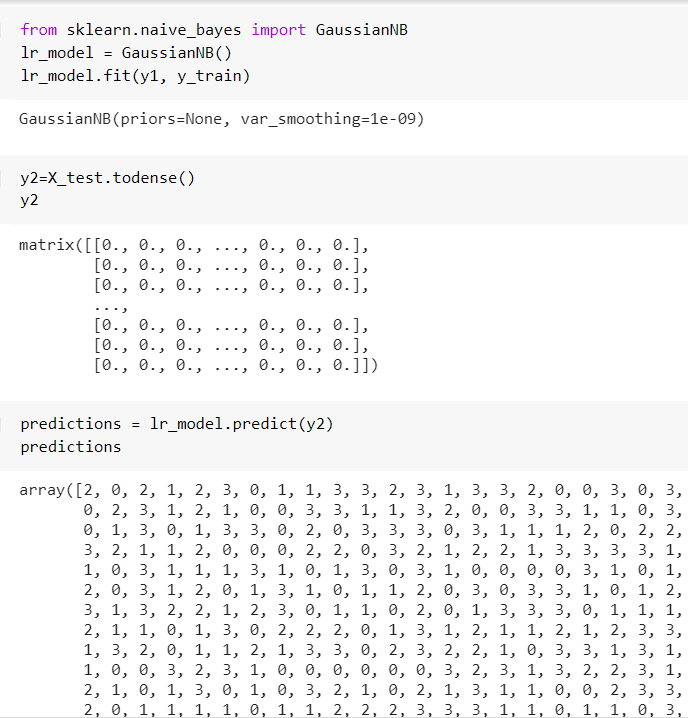


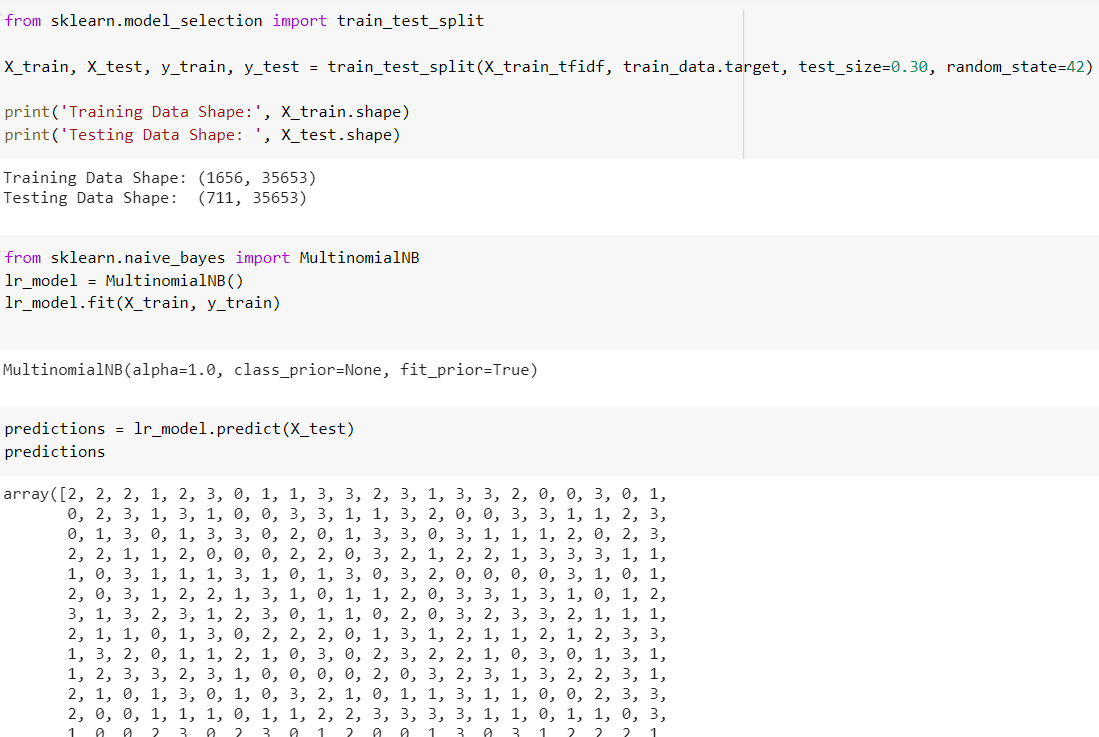


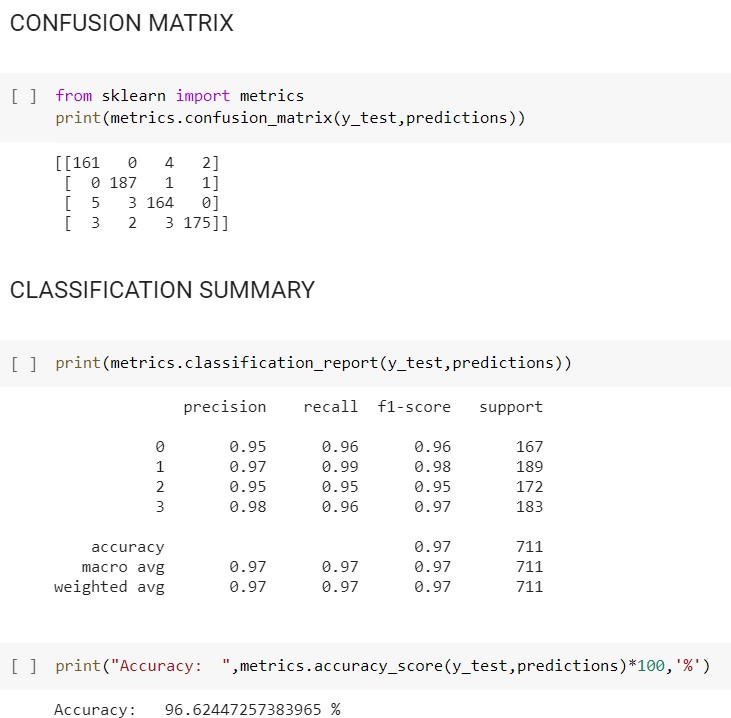










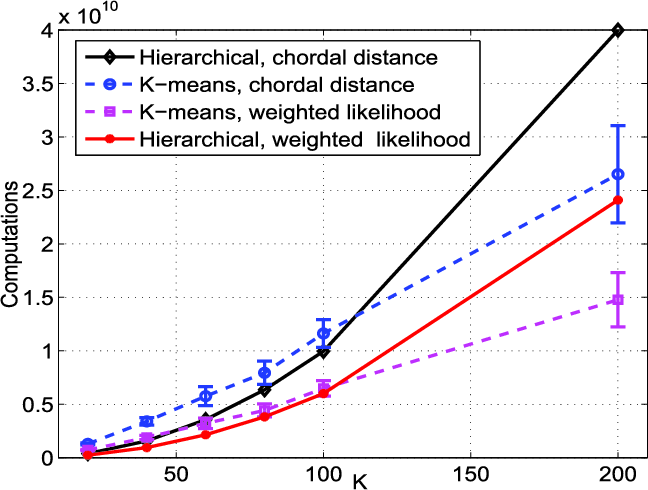


**Result: Multinomial Naïve Bayes (96 %) >Naïve Bayes (90%) > KNN(82%)**

# Logo Description automatically generated with medium confidenceEXPERIMENT NO. 8 TWIA Lab Manual (CSL 554)

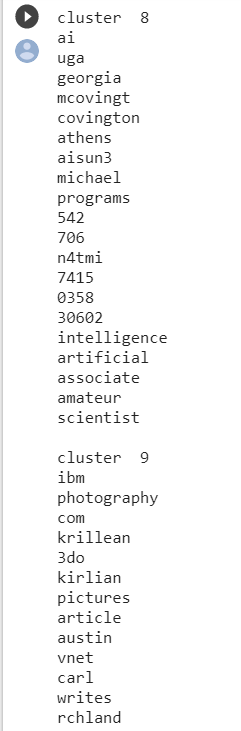
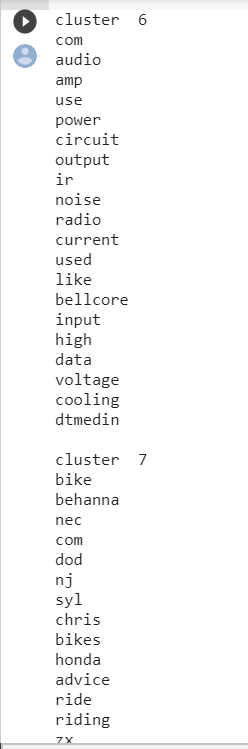
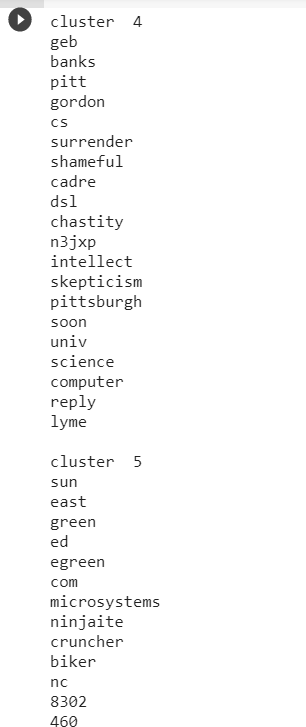
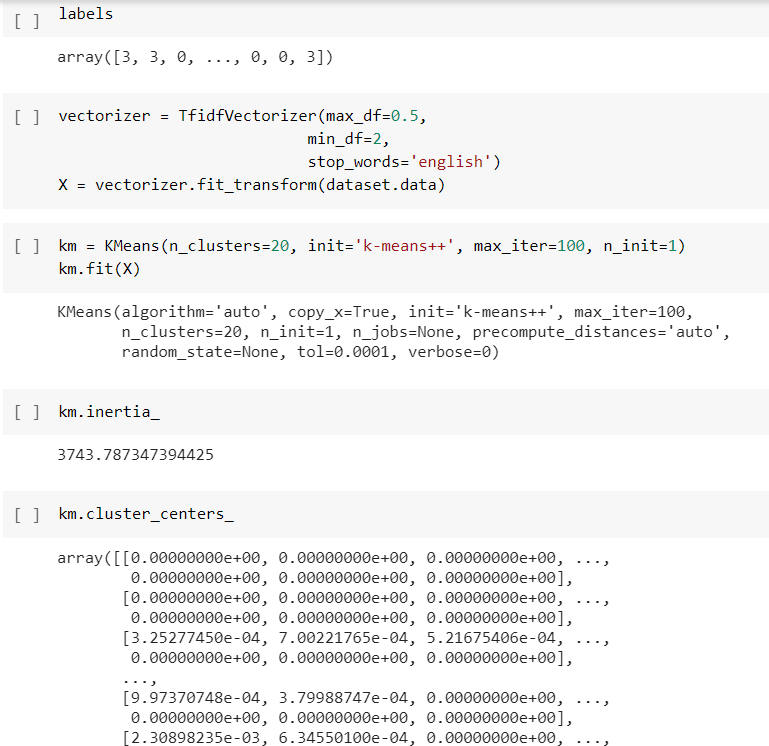
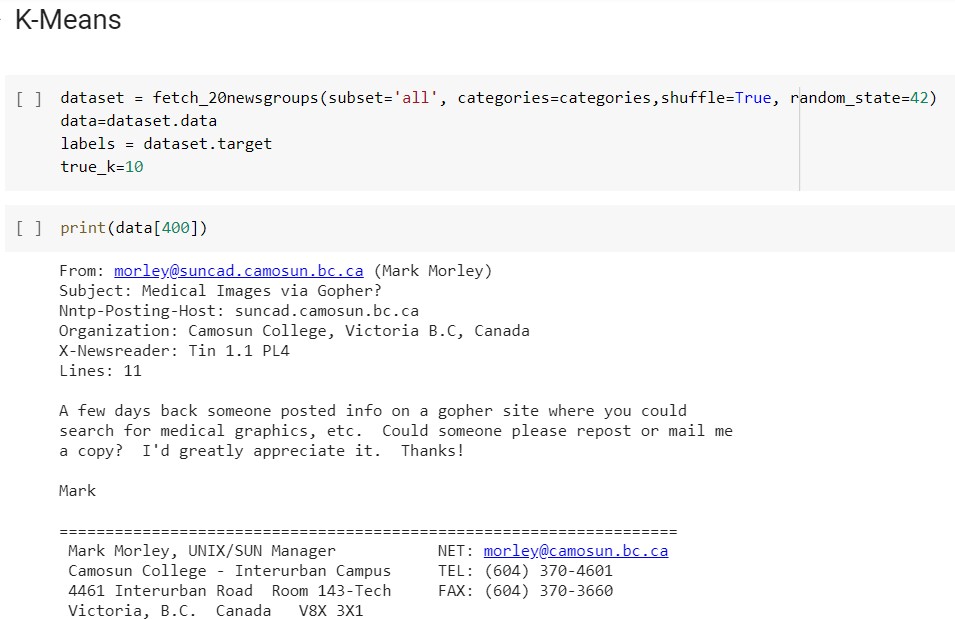
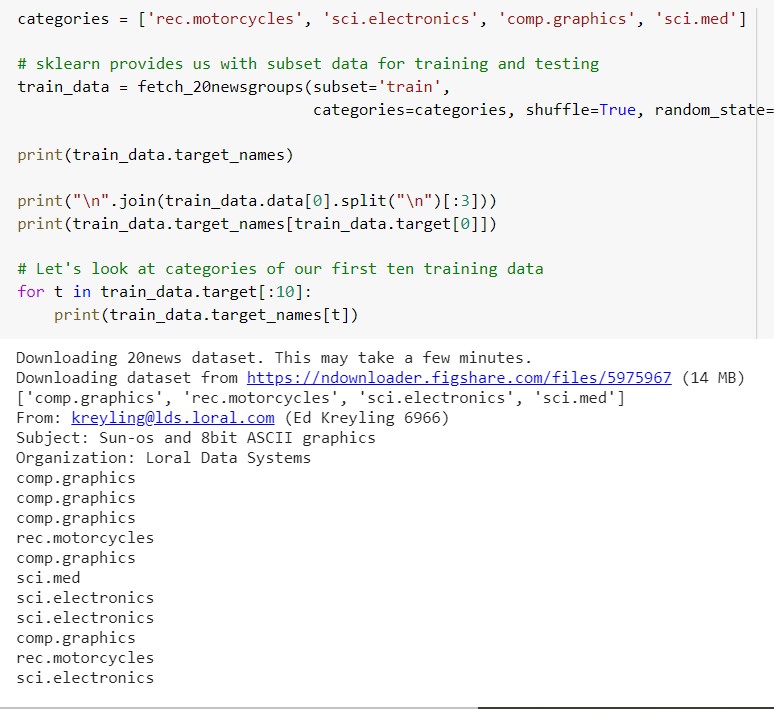
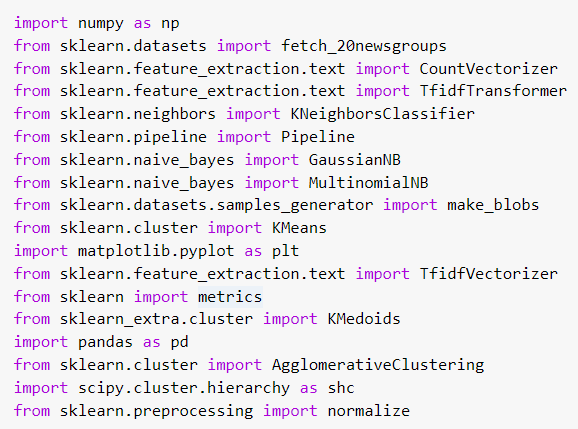
**2020-21**

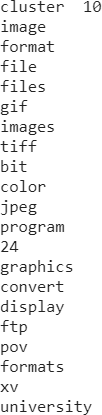
|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 2 May 2021** |
| **Faculty Signature:** |
| **Grade:** |
| **Objectives:**  **1.** Implementation of K Means, K Medoids and Hierarchical Clustering algorithms on text data.  **Background Study:**  **K-Means**  K-Means Clustering is an [unsupervised Learning algorithm](https://www.javatpoint.com/unsupervised-machine-learning), which groups the unlabeled dataset into different clusters in such a way that each dataset belongs to the group which has similar properties. Here K defines the number of pre-defined clusters that need to be created in the process.  It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.  **K-Medoids**  K*-*medoids chooses actual data points as centers ([medoids](https://en.wikipedia.org/wiki/Medoids) or exemplars), and thereby allows for greater interpretability of the cluster centers than in *k*-means, where the center of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster). Furthermore, *k*-medoids can be used with arbitrary dissimilarity measures, whereas *k*-means generally requires [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance) for efficient solutions*.*  **Hierarchical Clustering**  Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other. |

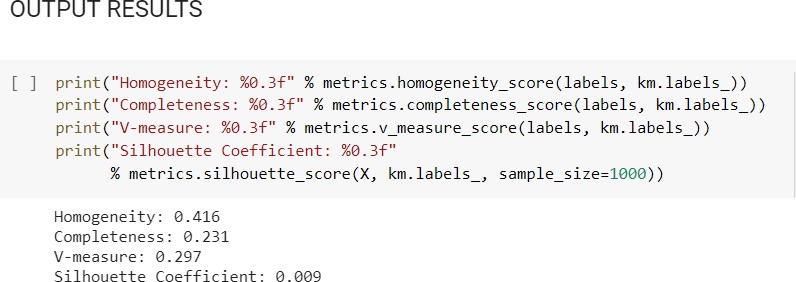


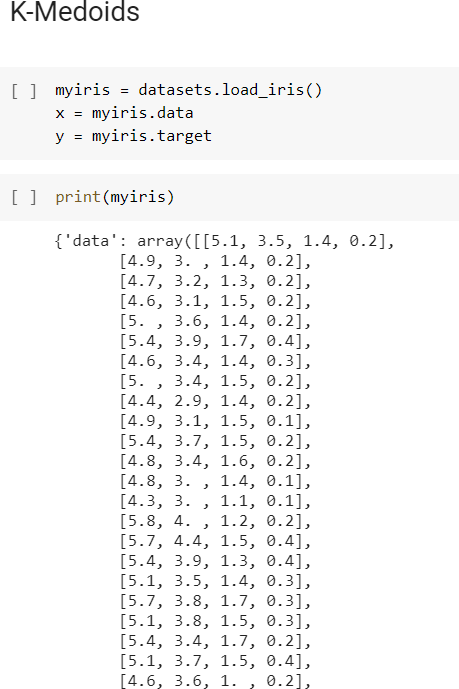
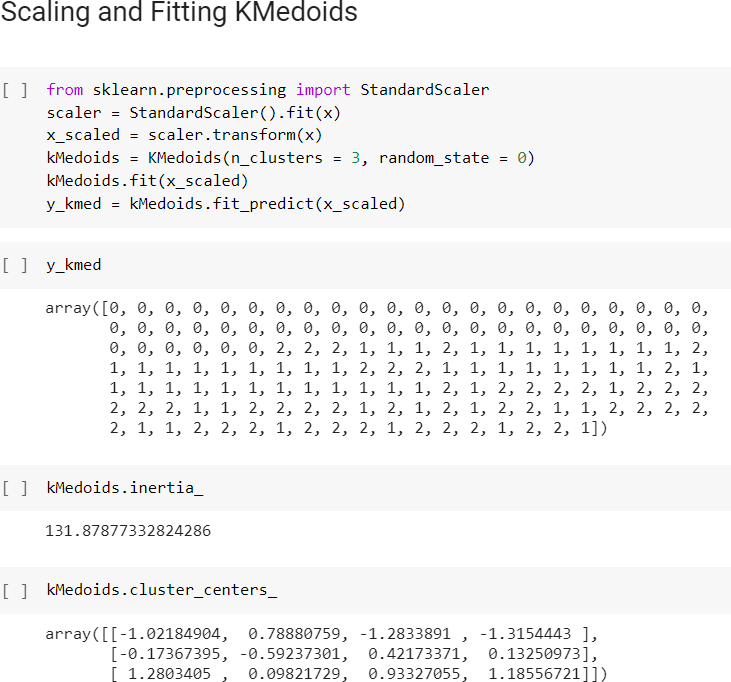
**Outcome:** Student will be able to differentiate between the three clustering algorithms and compare their accuracies. **Problem Statement:** Implement K-Means, K-Medoids and Hierarchical Clustering Algorithms on text data and plot visualizations for the same.

# CODE AND OUTPUT:

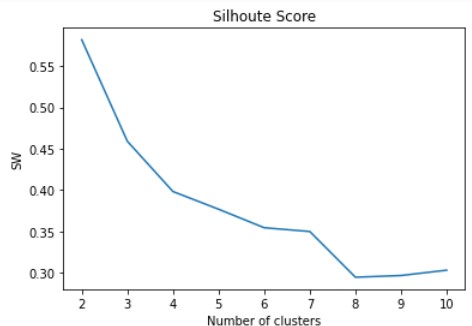
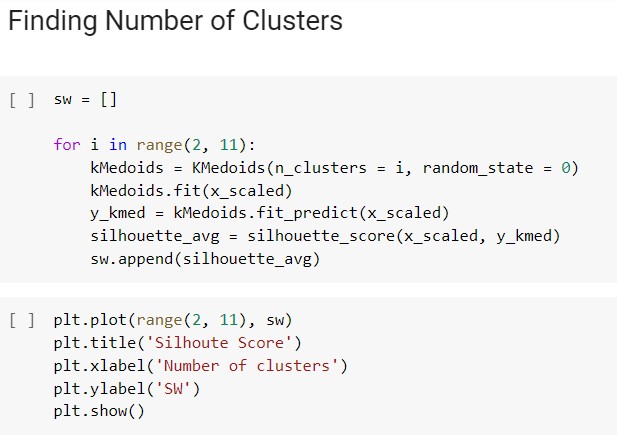




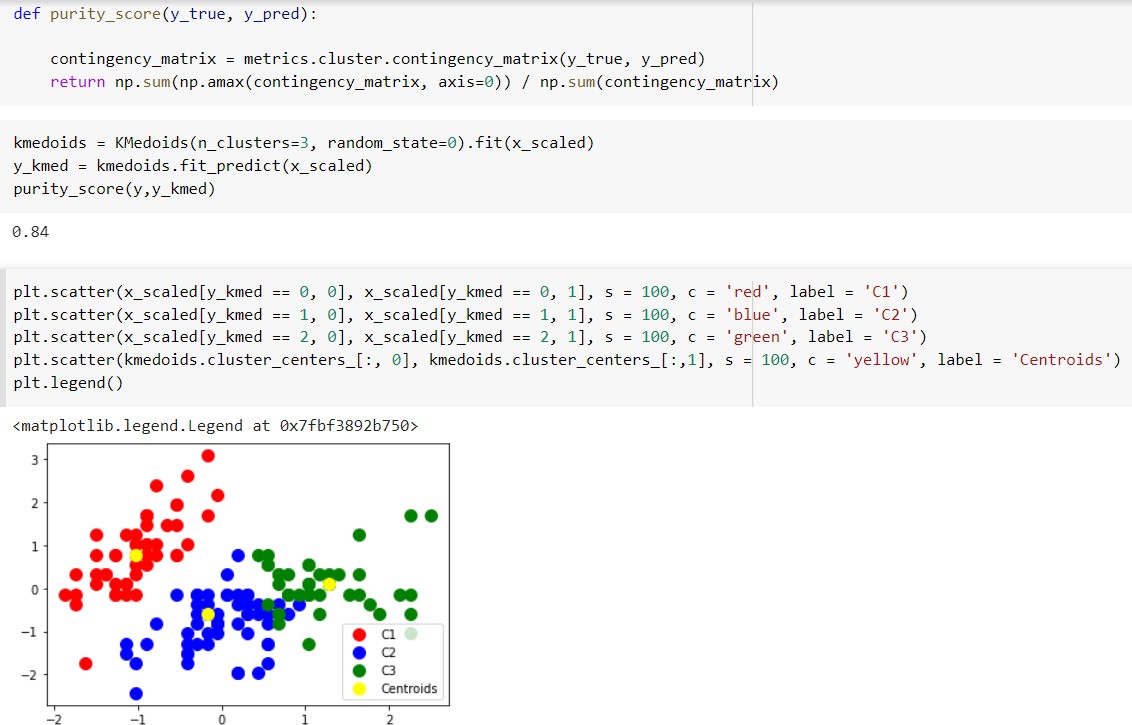


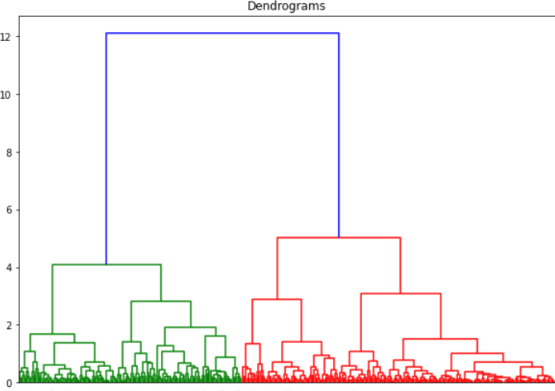


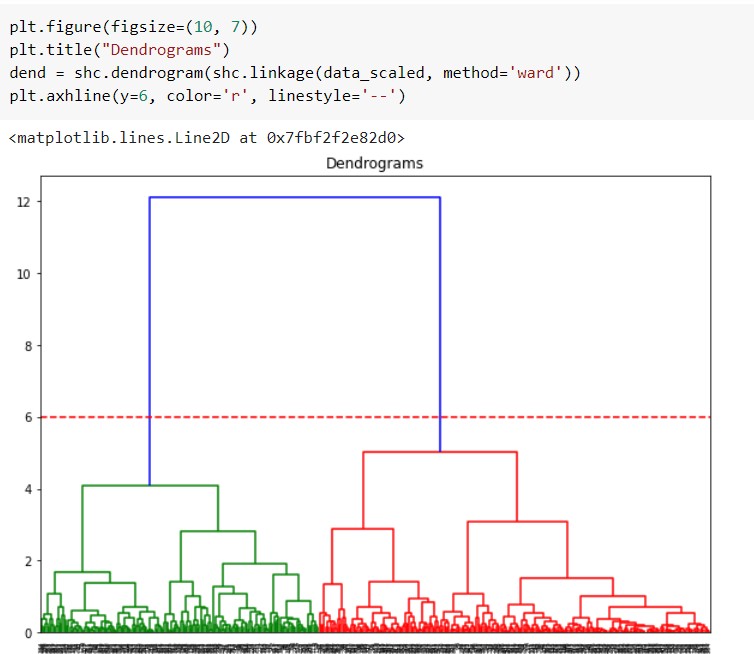


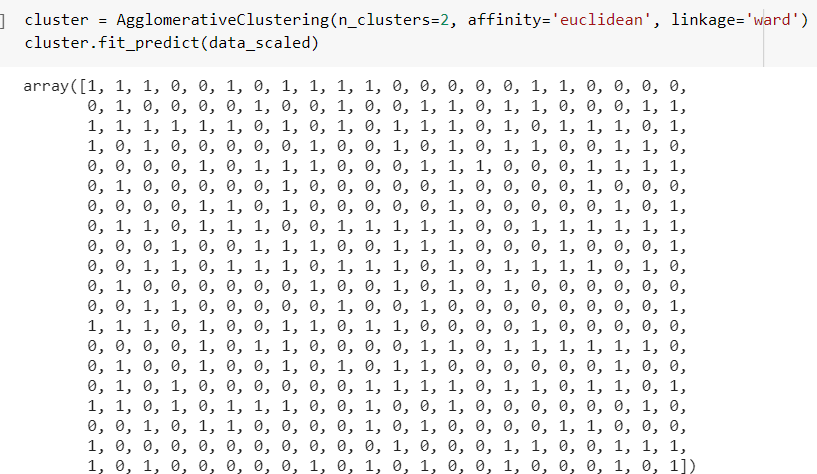


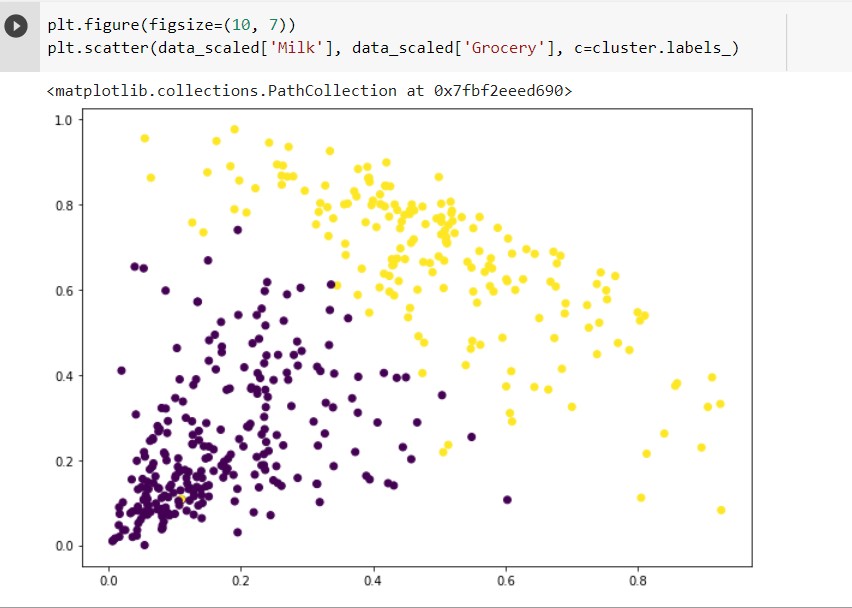
**Purity Score**













|  |
| --- |
| **TWIA Lab Manual (CSL 554)**  **EXPERIMENT NO. 9 2020-21** |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 9 May 2021** |
| **Faculty Signature:** |
| **Grade:** |

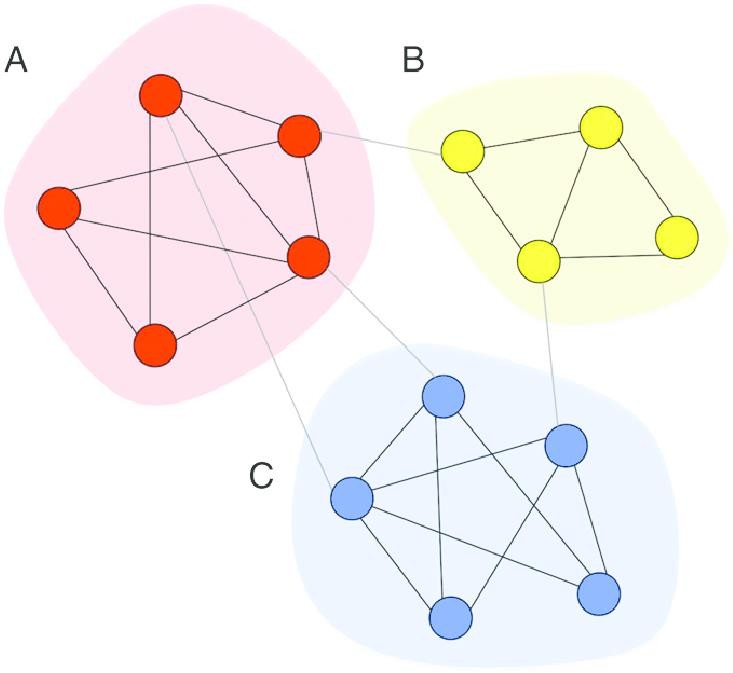
# Objectives:

* 1. Implement homophily for social media analysis.

# Background Study:

**Homophily**

Network homophily refers to the theory in [network science](https://en.wikipedia.org/wiki/Network_science) which states that, based on node attributes, similar nodes may be more likely to attach to each other than dissimilar ones. The hypothesis is linked to the model of [preferential attachment](https://en.wikipedia.org/wiki/Preferential_attachment) and it draws from the phenomenon of homophily in social sciences and much of the scientific analysis of the creation of social ties based on similarity comes from network science. In fact, empirical research seems to indicate the frequent occurrence of homophily in real networks.



**Outcome:** Students will be able to perform a case study on Social Network Analysis for a population.

# Problem Statement:

In this case study, you will investigate homophily of several characteristics of individuals connected in social networks in rural India.

You will calculate the chance homophily for an arbitrary characteristic. Homophily is the proportion of edges in the network whose constituent nodes share that characteristic. How much homophily do we expect by chance? If characteristics are distributed completely randomly, the probability that two nodes x and y share characteristic a is the probability both nodes have characteristic a, which is the frequency of a squared. The total probability that

nodes x and y share their characteristic is therefore the sum of the frequency of each characteristic in the network.

# DATA CAMP CASE STUDY QUESTIONS AND ANSWERS

Q1. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will calculate the chance homophily for an arbitrary characteristic. Homophily is the proportion of edges in the network whose constituent nodes share that characteristic. How much homophily do we expect by chance? If characteristics are distributed completely randomly, the probability that two

nodes x and y share characteristic a is the probability both nodes have characteristic a, which is the frequency of a squared. The total probability that nodes x and y share their characteristic is therefore the sum of the frequency of each characteristic in the network. For example, in the dictionary favorite\_colors provided, the frequency of red and blue is 1/3 and 2/3 respectively, so the chance homophily is (1/3)^2+(2/3)^2 = 5/9.

**100 XP**

* Create a function that takes a dictionary chars with personal IDs as keys and characteristics as values, and returns a dictionary with characteristics as keys, and the frequency of their occurrence as values.
* Create a function chance\_homophily(chars) that takes a dictionary chars defined as above and computes the chance homophily for that characteristic.
* A sample of three peoples' favorite colors is given in favorite\_colors. Use your function to compute the chance homophily in this group, and store as color\_homophily.
* Print color\_homophily.

CODE:

from collections import Counter def chance\_homophily(chars):

"""

Computes the chance homophily of a characteristic, specified as a dictionary, chars.

"""

#enter your code here

chars\_counts\_dict = Counter(chars.values())

chars\_counts = np.array(list(chars\_counts\_dict.values())) chars\_props = chars\_counts / sum(chars\_counts)

return sum(chars\_props\*\*2)

favorite\_colors = { "ankit": "red",

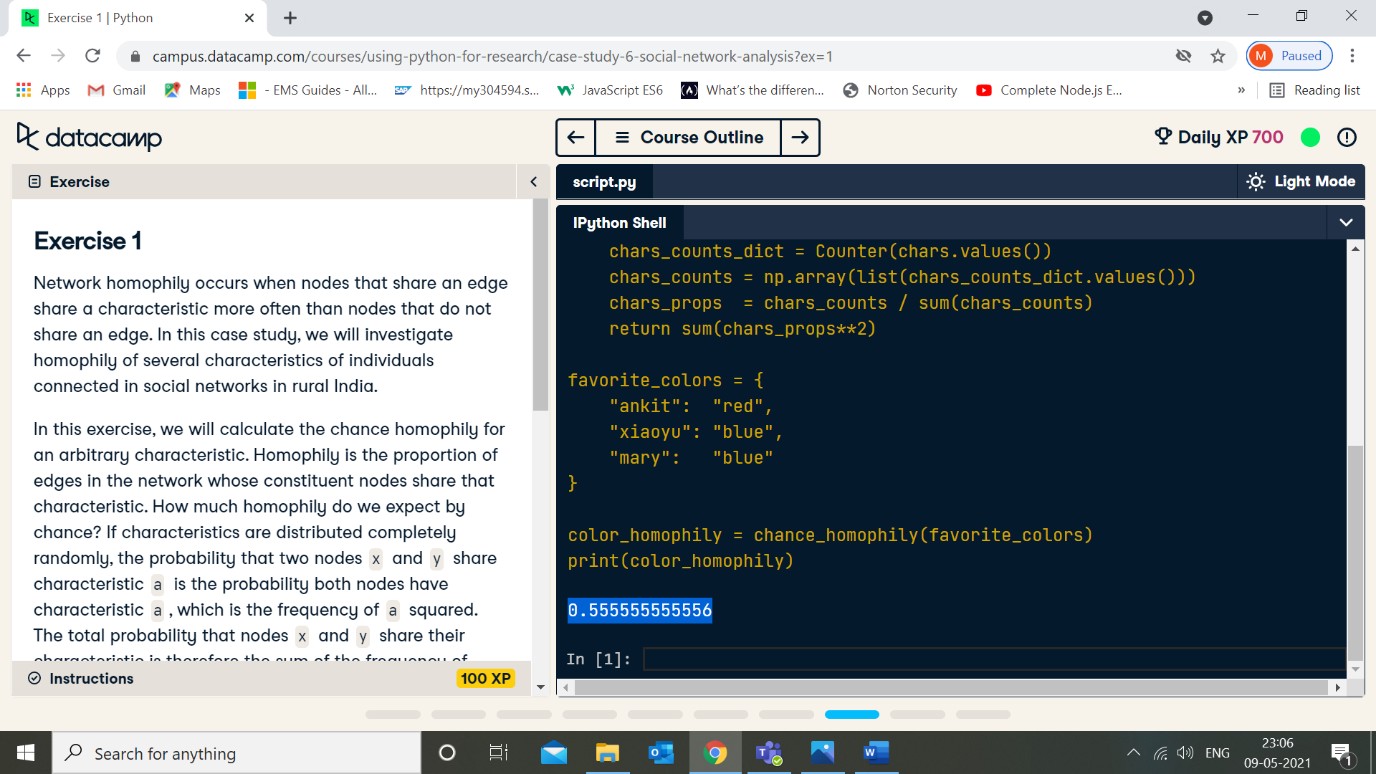
"xiaoyu": "blue",

"mary": "blue"

}

color\_homophily = chance\_homophily(favorite\_colors) print(color\_homophily)

OUTPUT:



Q2. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In the remaining exercises, we will calculate and compare the actual homophily in these village to chance. In

this exercise, we subset the data into individual villages and store them.

**Instructions**

**100 XP**

* individual\_characteristics.dta contains several characteristics for each individual in the dataset such as age, religion, and caste. Use the pandas library to read in and store these characteristics as a dataframe called df.
* Store separate datasets for individuals belonging to Villages 1 and 2 as df1 and df2, respectively.

o Note that some attributes may be missing for some individuals. In this case study, we will ignore rows of data where some column information is missing.

* Use the head method to display the first few entries of df1.

CODE :

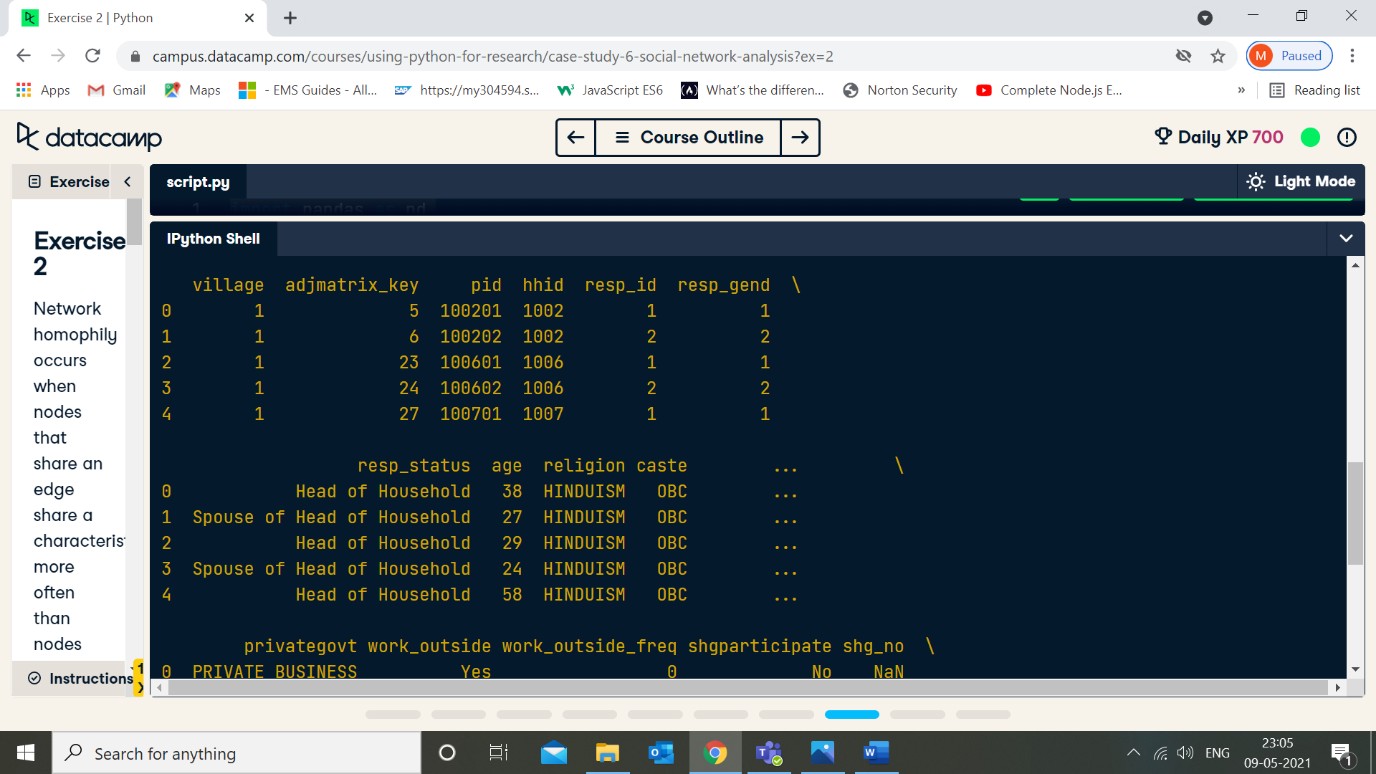
import pandas as pd

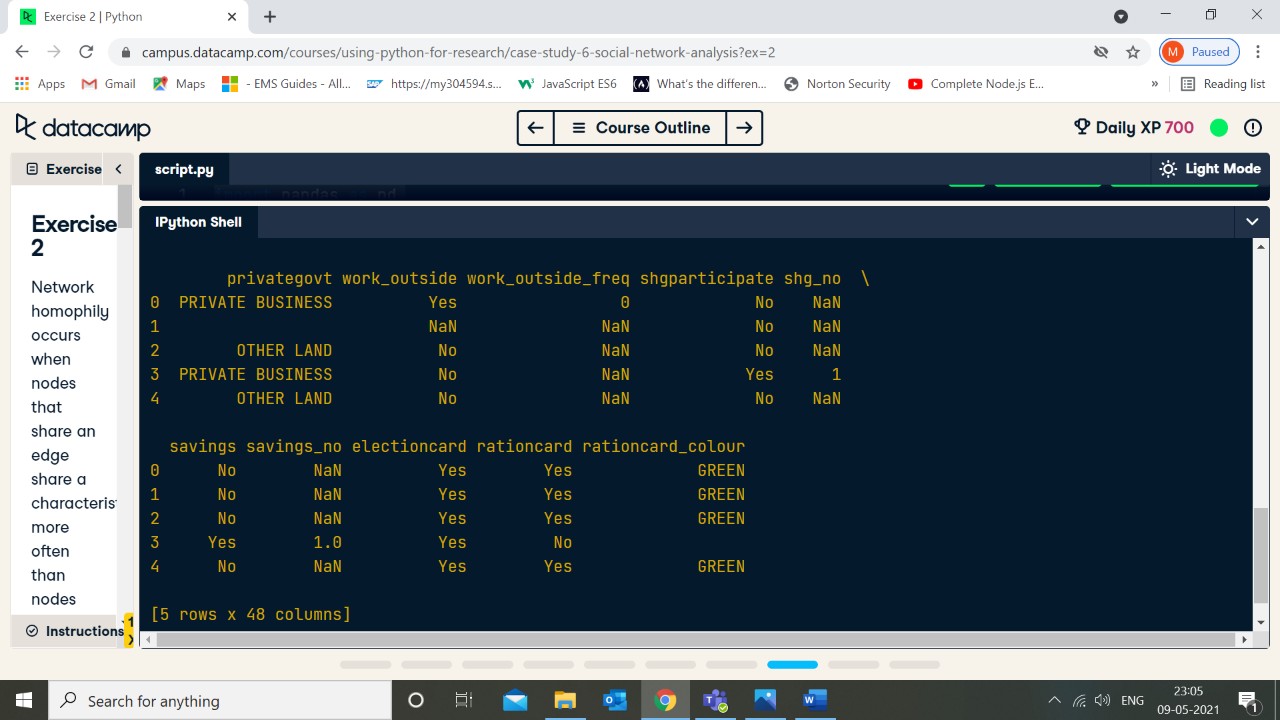
df = pd.read\_stata(data\_filepath + "individual\_characteristics.dta") df1 = df[df["village"]==1]# Enter code here!

df2 = df[df["village"]==2]# Enter code here!

# Enter code here! df1.head()

OUTPUT:





Q3. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we define a few dictionaries that enable us to look up the sex, caste, and religion of members of each village by personal ID. For Villages 1 and 2, their personal IDs are stored as pid.

**Instructions**

**100 XP**

* Define dictionaries with personal IDs as keys and a given covariate for that individual as values. Complete this for the sex, caste, and religion covariates, for Villages 1 and 2.
* For Village 1, store these dictionaries into variables named sex1, caste1, and religion1.
* For Village 2, store these dictionaries into variables named sex2, caste2, and religion2.

CODE:

sex1

caste1

= df1.set\_index("pid")["resp\_gend"].to\_dict()

= df1.set\_index("pid")["caste"].to\_dict()

religion1 = df1.set\_index("pid")["religion"].to\_dict()

sex2

caste2

= df2.set\_index("pid")["resp\_gend"].to\_dict()

= df2.set\_index("pid")["caste"].to\_dict()

religion2 = df2.set\_index("pid")["religion"].to\_dict()

Q4. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will print the chance homophily of several characteristics of Villages 1 and 2. The function chance\_homophily is still defined from Exercise 1.

**Instructions**

**100 XP**

* sex1, caste1, religion1, sex2, caste2, and religion2 are already defined from previous exercises.

Use chance\_homophily to compute the chance homophily for sex, caste, and religion In Villages 1 and 2. Is the chance homophily for any attribute very high for either village?

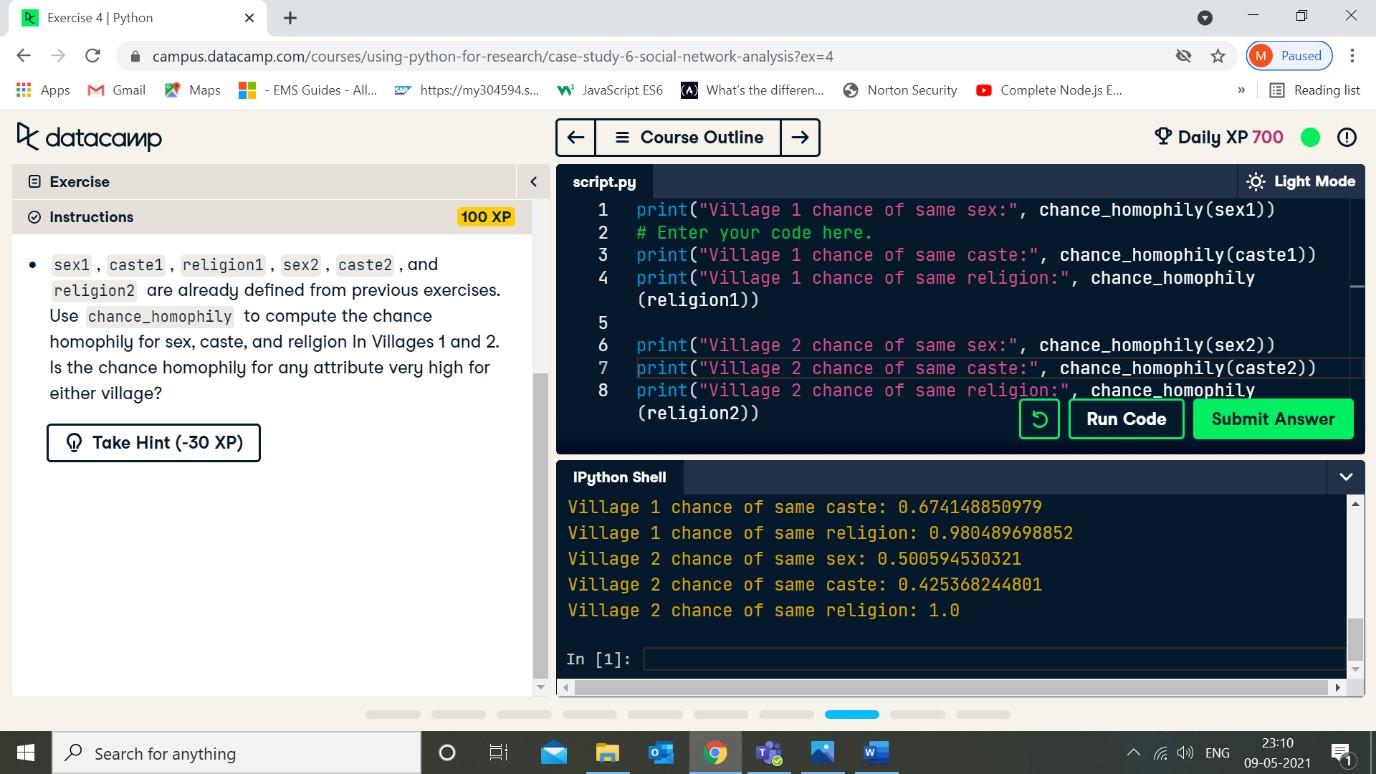
CODE :

print("Village 1 chance of same sex:", chance\_homophily(sex1)) # Enter your code here.

print("Village 1 chance of same caste:", chance\_homophily(caste1)) print("Village 1 chance of same religion:", chance\_homophily(religion1))

print("Village 2 chance of same sex:", chance\_homophily(sex2)) print("Village 2 chance of same caste:", chance\_homophily(caste2)) print("Village 2 chance of same religion:", chance\_homophily(religion2))

OUTPUT:



Q5. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will create a function that computes the observed homophily given a village and characteristic.

**Instructions**

**100 XP**

* Complete the function homophily(), which takes a network G, a dictionary of characteristics chars, and node IDs IDs. For each node pair, determine whether a tie exists between them, as well as whether they share a characteristic. The total count of these is num\_same\_ties and num\_ties respectively, and their ratio is the homophily of chars in G. Complete the function by choosing where to

increment num\_same\_ties and num\_ties.

CODE:

def homophily(G, chars, IDs): num\_same\_ties, num\_ties = 0, 0 for n1 in G.nodes():

for n2 in G.nodes():

if n1 > n2: # do not double-count edges! if IDs[n1] in chars and IDs[n2] in chars:

\_same\_ties`?

if G.has\_edge(n1, n2):

num\_ties += 1# Should `num\_ties` be incremented? What about `num

if chars[IDs[n1]] == chars[IDs[n2]]:

num\_same\_ties += 1# Should `num\_ties` be incremented? What a

bout `num\_same\_ties`?

return (num\_same\_ties / num\_ties)

Q6. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will obtain the personal IDs for Villages 1 and 2. These will be used in the next exercise to calculate homophily for these villages.

**Instructions**

**100 XP**

* In this dataset, each individual has a personal ID, or PID, stored in key\_vilno\_1.csv and key\_vilno\_2.csv for villages 1 and 2, respectively. data\_filepath contains the base URL to the datasets used in this exercise. Use pd.read\_csv to read in and store key\_vilno\_1.csv and key\_vilno\_2.csv as pid1 and pid2 respectively.

The csv files have no headers, so make sure to include the parameter header = None.

CODE:

pid1 = pd.read\_csv(data\_filepath + "key\_vilno\_1.csv", dtype=int, header = None) pid2 = pd.read\_csv(data\_filepath + "key\_vilno\_2.csv", dtype=int, header = None)

Q7. Network homophily occurs when nodes that share an edge share a characteristic more often than nodes that do not share an edge. In this case study, we will investigate homophily of several characteristics of individuals connected in social networks in rural India.

In this exercise, we will compute the homophily of several network characteristics for Villages 1 and 2, and compare this to chance homophily. The networks for these villages have been stored as networkx graph objects G1 and G2. homophily() and chance\_homophily() are pre-loaded from previous exercises.

CODE :

print("Village 1 observed proportion of same sex:", homophily(G1, sex1, pid1)) # Enter your code here!

print("Village 1 observed proportion of same caste:", homophily(G1, caste1, pid1)) print("Village 1 observed proportion of same religion:", homophily(G1, religion1, pid1))

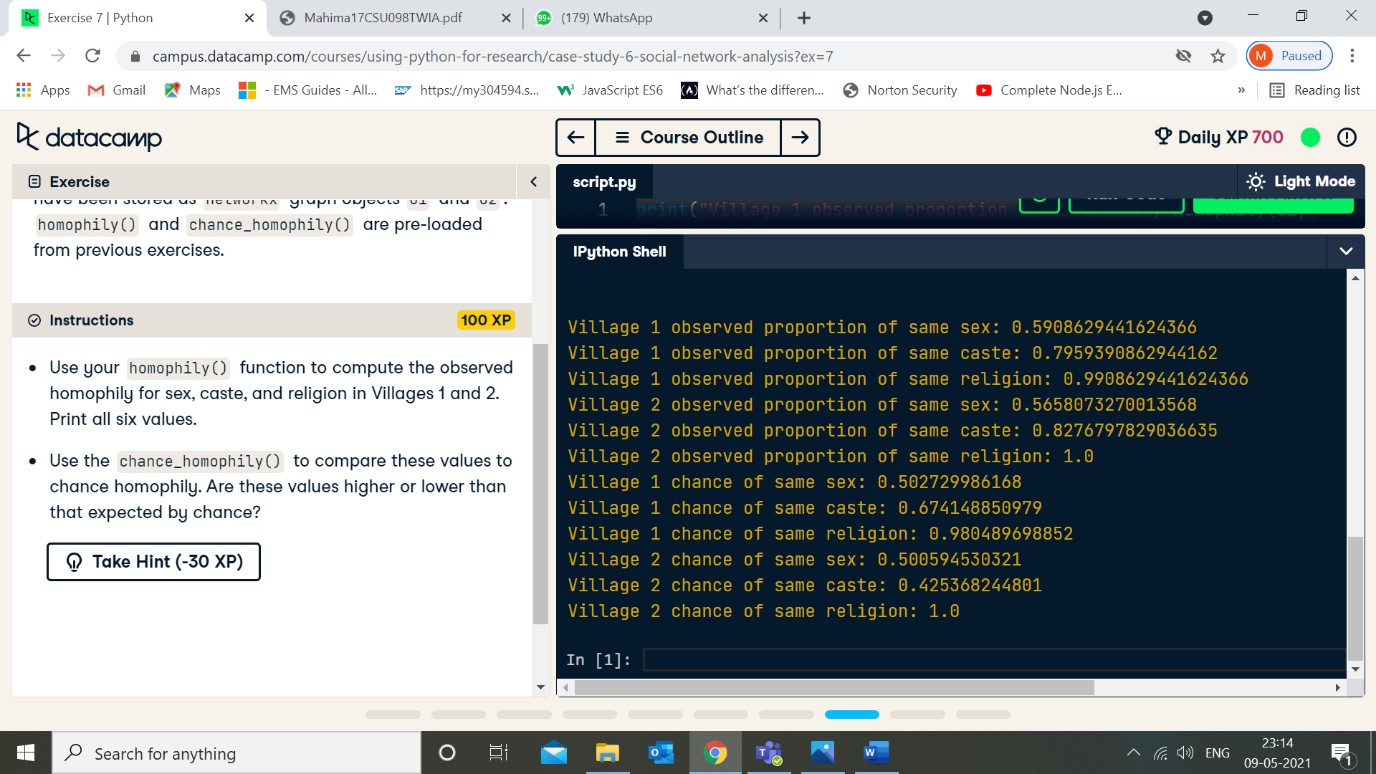
print("Village 2 observed proportion of same sex:", homophily(G2, sex2, pid2)) print("Village 2 observed proportion of same caste:", homophily(G2, caste2, pid2)) print("Village 2 observed proportion of same religion:", homophily(G2, religion2, pid2))

print("Village 1 chance of same sex:", chance\_homophily(sex1))

print("Village 1 chance of same caste:", chance\_homophily(caste1)) print("Village 1 chance of same religion:", chance\_homophily(religion1))

print("Village 2 chance of same sex:", chance\_homophily(sex2)) print("Village 2 chance of same caste:", chance\_homophily(caste2)) print("Village 2 chance of same religion:", chance\_homophily(religion2))

OUTPUT:



# VALUE ADDED EXPERIMENT NO. 1

**TWIA Lab Manual (CSL 554)**

# 2020-21

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 3 March 2021** |
| **Faculty Signature:** |
| **Grade:** |

**Objective:**

# Apply chunking on a sentence and explore text corpora.

1. **Apply chinking on a piece of text.**

# Implementation of Named Entity recognition.

**Background Study:**

# Chunking

Chunking refers to the process of taking individual pieces of information and grouping them into larger units. By grouping each data point into a larger whole, we can improve the amount of information we can remember.

# Chinking

Chinking is the process of removing a sequence of tokens from a chunk. If the matching sequence of tokens spans an entire chunk, then the whole chunk is removed; if the sequence of tokens appears in the middle of the chunk, these tokens are removed, leaving two chunks where there was only one before. If the sequence is at the periphery of the chunk, these tokens are removed, and a smaller chunk remains.

# Named Entity

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on.

# Named Entity Recognition

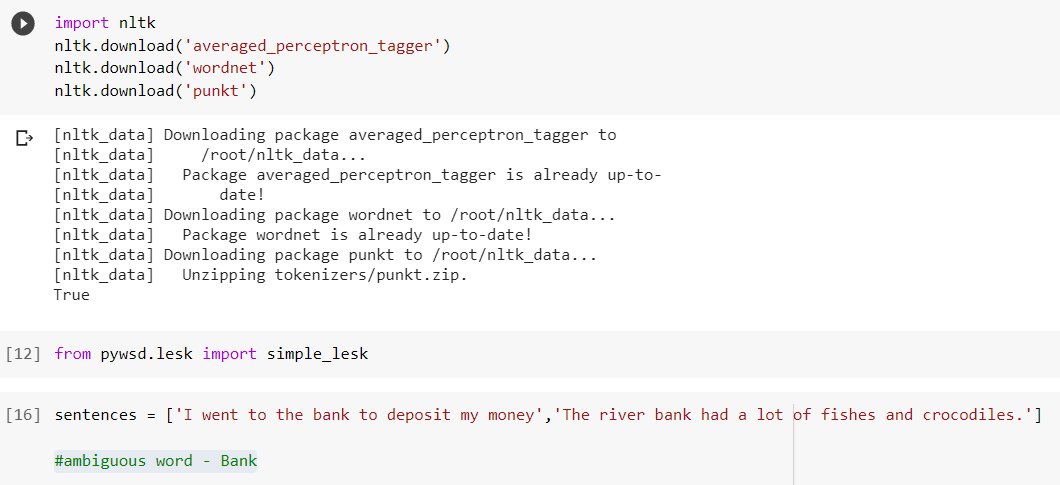
[Named entity recognition](https://en.wikipedia.org/wiki/Named-entity_recognition) (NER)is probably the first step towards information extraction that seeks to locate and classify [named entities](https://en.wikipedia.org/wiki/Named_entity) in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

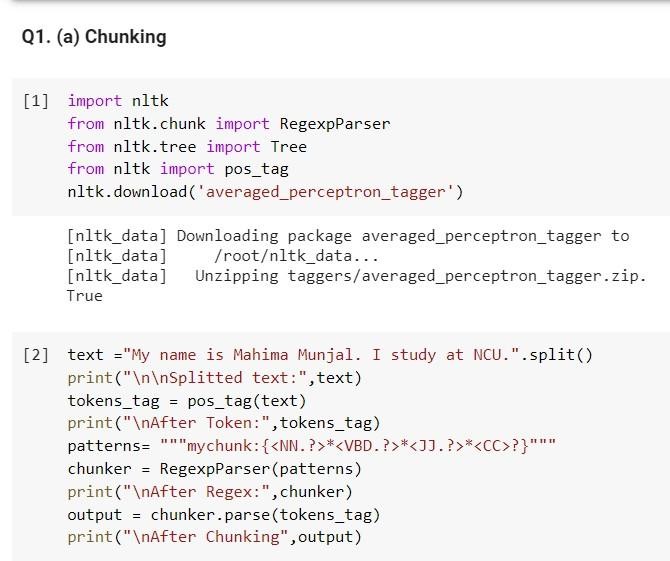
**Outcome:** Students will be able to understand the concept such as chunking, regex parser, chinking and tagging. They will also be able to explore the corpus brown and apply named entity recognition using ne\_chunk() functions and Spacy library.

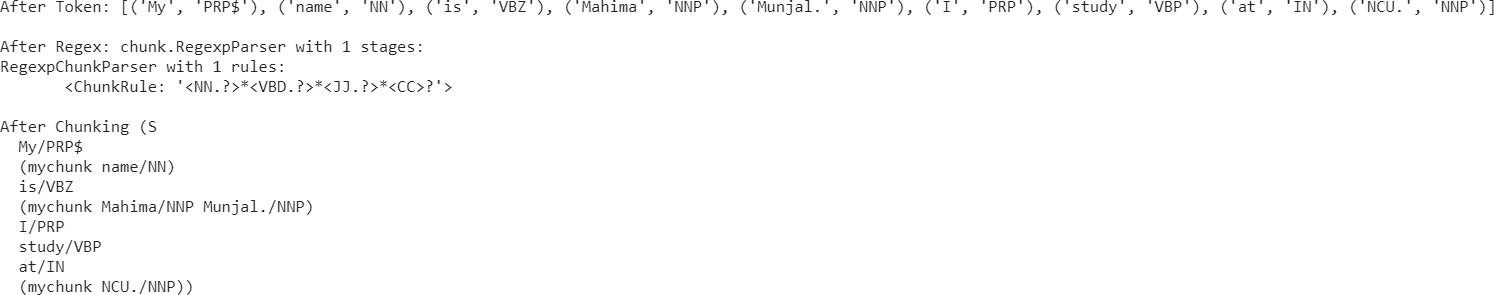
# Problem Statement:

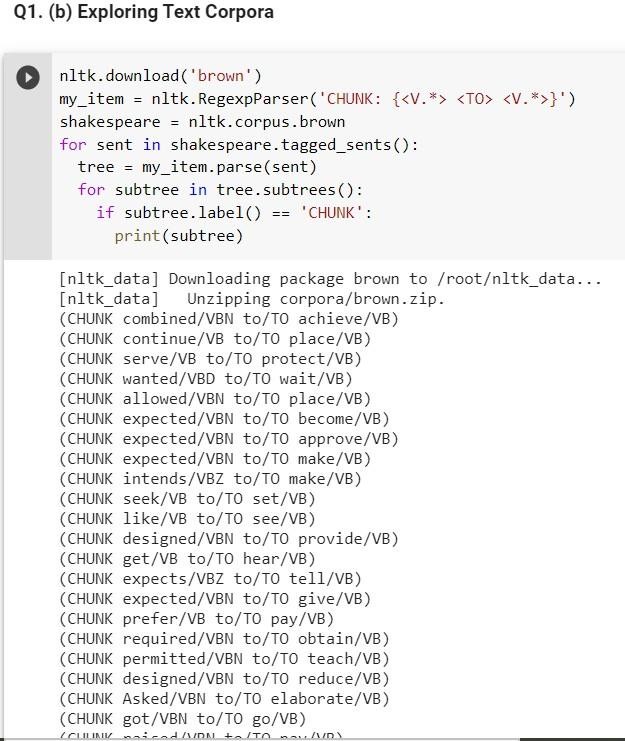
1. **Apply chunking on a sentence and explore text corpora.**

# CODE AND OUTPUT:



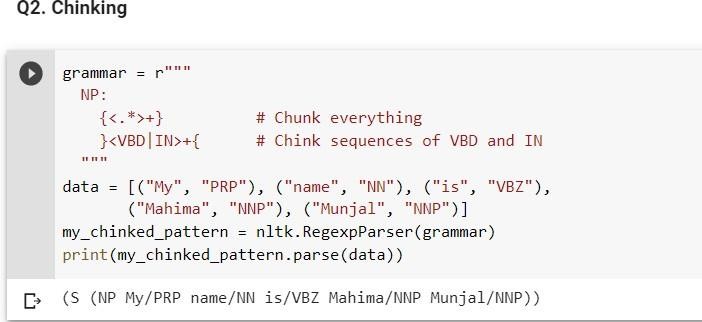






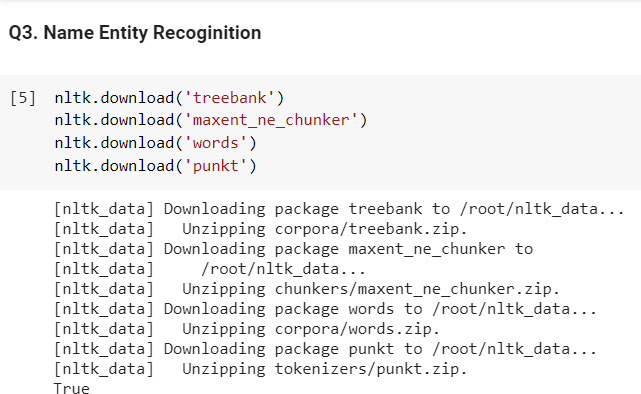
1. **Apply chinking on a piece of text.**

# CODE AND OUTPUT:

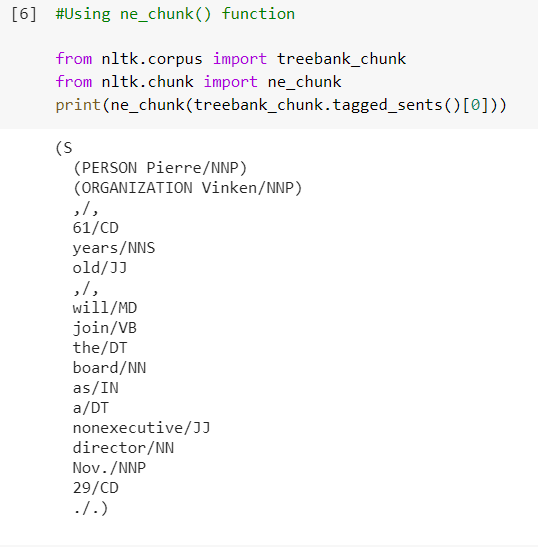


1. **Implementation of Named Entity recognition.**

# CODE AND OUTPUT:



1. **Using ne\_chunk( ) function**



# Using Spacy



**VALUE ADDED EXPERIMENT NO. 2**

# TWIA Lab Manual (CSL 554)

**2020-21**

|  |
| --- |
| **Student Name and Roll Number: MUSKAN NANDKANI (17CSU121)** |
| **Semester /Section: VIII** |
| **Link to Code:** [dheeraj063/Lab-Manual- (github.com)](https://github.com/dheeraj063/Lab-Manual-) |
| **Date: 22 January 2021** |
| **Faculty Signature:** |
| **Grade:** |

# Objective:

* 1. **Apply word and sentence tokenization on a piece of text.**

# Implement Punkt Tokenization

* 1. **Explore Gutenberg Corpus.**

# Background Study:

**Tokenization**

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

# Punkt Sentence Tokenizer

This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintexts in the target language before it can be used.

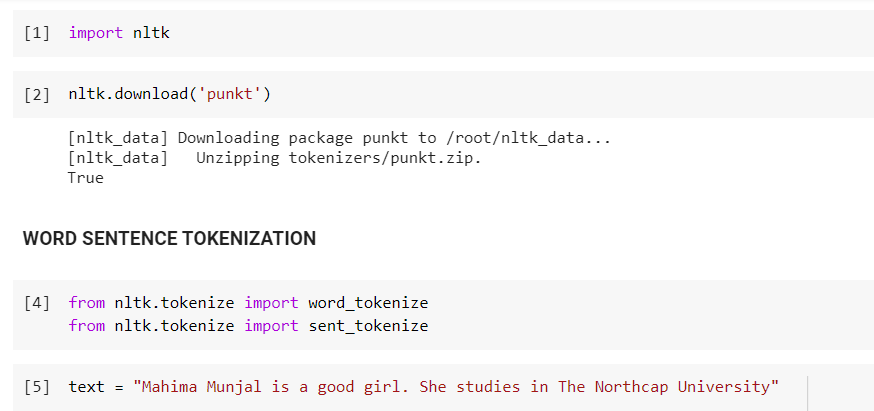
# Gutenberg Corpus

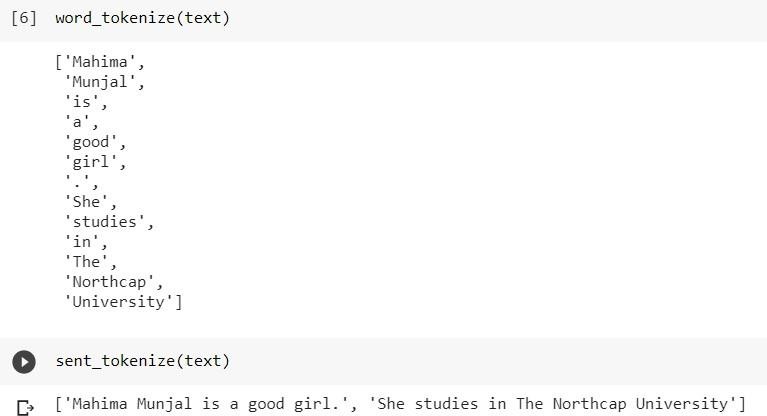
The Project Gutenberg English corpus is a corpus made up of all English e-books available in the [Gutenberg database](http://www.gutenberg.org/about/) in October 2014. Project Gutenberg Corpus, an open science approach to a curated version of the complete PG data containing more than 50,000 books and more than 3×109 word-tokens.

**Outcome:** Students will be able to understand the concept of tokenization and why it is an important aspect of NLTK and why it is needed. They will also be able to learn Punkt Tokenization. They will also be able to explore Gutenberg Corpus.

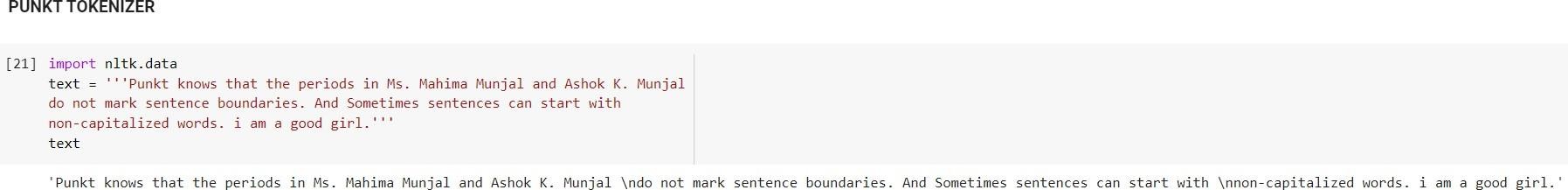
# Problem Statement:

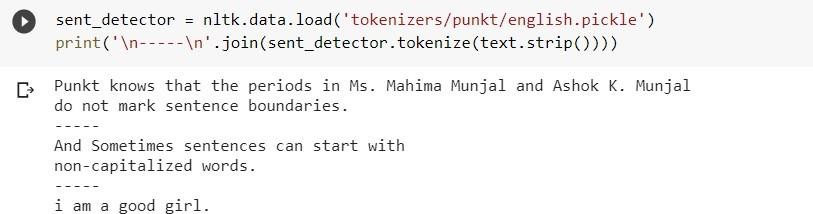
1. **Apply word and sentence tokenization on a piece of text. CODE AND OUTPUT:**





# Implement Punkt Tokenization CODE AND OUTPUT:





1. **Explore Gutenberg Corpus.**

# CODE AND OUTPUT:

