

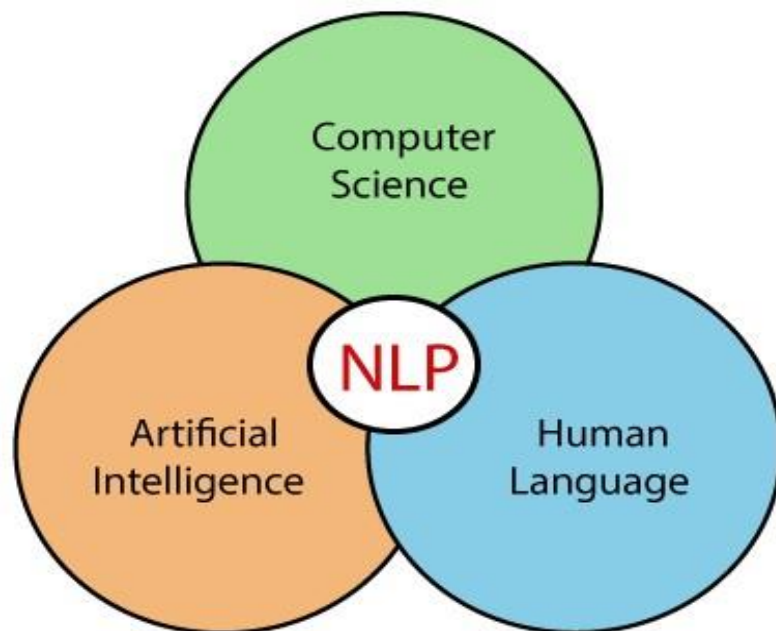
# NLP PROJECT

Final Report(Round 1 & Round 2)

By Group- TEAM ONE

Arin Khandelwal (21UCS027)    Devang Agarwal (21UCS057)

Pranvi Rai (21UCS157)        Ritik Goyal (21UCS171)



## Book Taken:-

Title : The Scam

Author : Debashis Basu, Sucheta Dalal

Book Link: [https://github.com/rtkgoyal/NLP-project\\_teamone/blob/main/the\\_scam.txt](https://github.com/rtkgoyal/NLP-project_teamone/blob/main/the_scam.txt)

## **INTRODUCTION**

Natural Language Processing (NLP) is a field of artificial intelligence that studies the interaction between computers and human language. It enables machines to understand, interpret, and generate human language, making it a pivotal technology in applications like chatbots, language translation, sentiment analysis, and text summarization. NLP uses algorithms and linguistic rules to process and analyze text data, extracting insights and facilitating communication between humans and machines. It plays a crucial role in tasks such as speech recognition and language understanding, and continues to advance, with applications in various industries, including healthcare, finance, and customer service.

## **OBJECTIVE**

The objective of the tasks is to perform comprehensive text analysis on a given book (T in TXT format) by implementing various NLP techniques.

In the first part, This includes preprocessing the text by removing non-essential elements, tokenizing and eliminating stop words, examining token frequency distributions, creating a word cloud, conducting PoS tagging, generating a bigram probability table for the largest chapter (C), and evaluating the accuracy of the bi-gram-based sentence completion for another chapter. These tasks aim to explore the linguistic structure and content of the book, demonstrating practical NLP applications and language analysis capabilities.

In the second part, the objective is twofold. First, recognize and categorize entities in the book using performance measures and manual labeling for evaluation. Second, generate TF-IDF vectors for individual chapters and measure their similarity, visualizing the results in a gradient table. The goal is to assess entity recognition performance and explore chapter-wise textual similarities using TF-IDF vector.

# Project Round 1

## Importing Libraries

Here, The provided code imports necessary libraries for text analysis, including NLTK, Matplotlib, and WordCloud. It aims to process and visualize text data, likely for tasks such as text mining and generating word clouds.

```
import re
import nltk
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from wordcloud import WordCloud
from collections import defaultdict
```

## Data Pre-Processing

The given code defines a preprocess\_text function to clean and preprocess text data by removing chapter headings, newlines, and vertical bars, likely for text analysis or natural language processing tasks.

```
# Function to perform simple text preprocessing (remove chapter headings, images, tables, etc.)
def preprocess_text(text):
    text = re.sub(r'Chapter \d+', '', text)
    text = re.sub(r'
                \n
                Image :.*?
                ', '', text)
    text = re.sub(r'\|', '', text)
    return text
```

## Tokenization of Data Set

The code defines a function `tokenize_and_remove_stopwords` that takes a text as input, tokenizes it, and removes common English stopwords using NLTK. It then preprocesses a novel's text using an undefined `preprocess_text` function and applies this tokenization and stopwords removal to it.

```
# Function to tokenize text and remove stopwords
def tokenize_and_remove_stopwords(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
    return filtered_tokens

text = preprocess_text(novel_text)
tokens = tokenize_and_remove_stopwords(text)
```

```
nltk_data] Downloading package stopwords to /root/nltk_data...
nltk_data] Package stopwords is already up-to-date!
nltk_data] Downloading package punkt to /root/nltk_data...
nltk_data] Package punkt is already up-to-date!
nltk_data] Downloading package averaged_perceptron_tagger to
nltk_data] /root/nltk_data...
nltk_data] Package averaged_perceptron_tagger is already up-to-date!
```

## Frequency Distribution of Tokens

The provided code computes the frequency distribution of tokens in a text and visualizes the 50 most common tokens using a bar chart. This analysis helps identify the most frequently occurring words in the text, allowing for insights into its content.

```
# Frequency distribution
fdist = FreqDist(tokens)

# Analysis part

print("Frequency Distribution of Tokens:")
print(fdist)

print("Most Common Words:")
print(fdist.most_common(10))

# Plotting the frequency distribution of 50 most common tokens

tokens, frequencies = zip(*fdist.most_common(50))

plt.figure(figsize=(18,8))
plt.bar(tokens, frequencies)

plt.xlabel("Token")
plt.ylabel("Frequency")
plt.title("Frequency Distribution of Tokens")

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

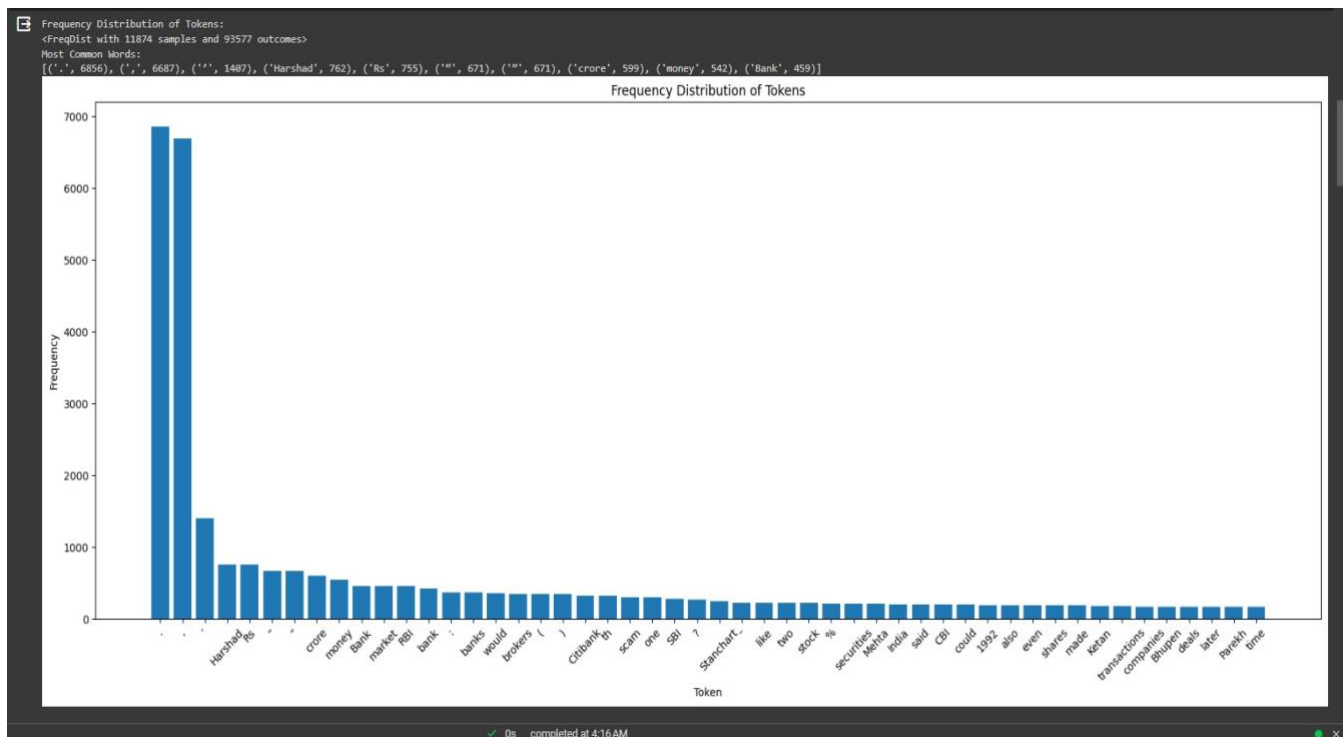
## Output:-

Frequency Distribution of Tokens:

<FreqDist with 11874 samples and 93577 outcomes>

Most Common Words:

[('.', 6856), (' ', 6687), (''', 1407), ('Harshad', 762), ('Rs', 755), ('"', 671), ('"', 671), ('crore', 599), ('money', 542), ('Bank', 459)]



## Creating Word Cloud

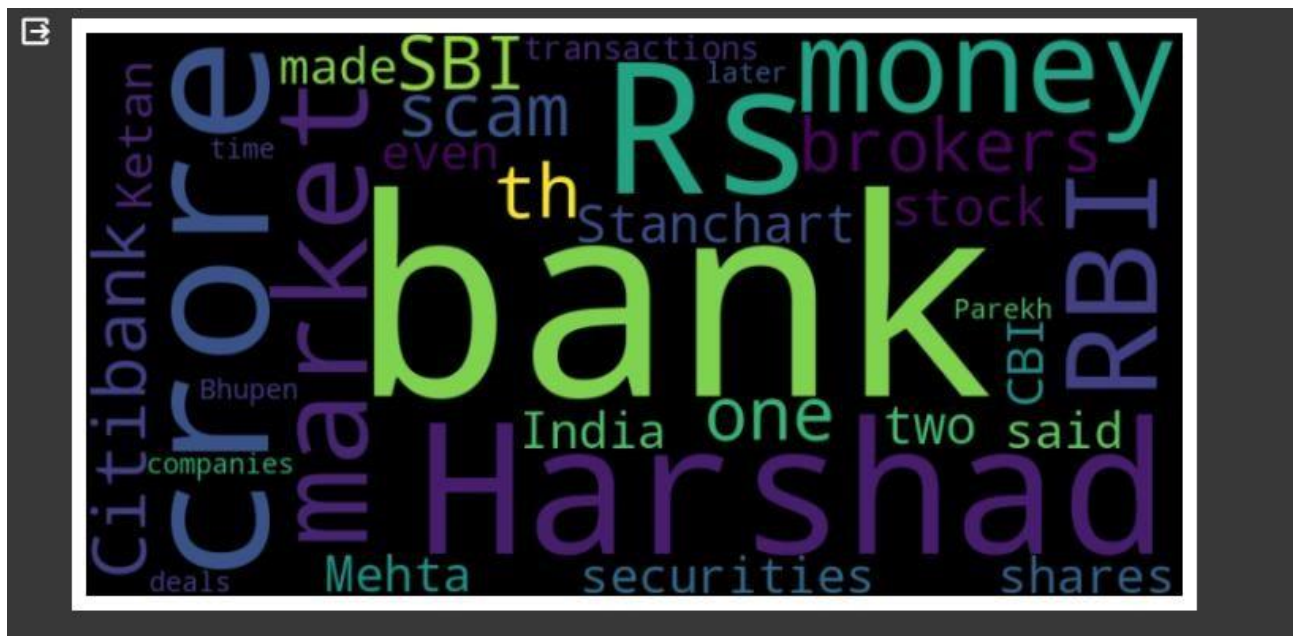
The code defines a function `create_word_cloud` that generates a word cloud from a list of tokens and saves it as an image. It then calls this function using the previously computed tokens, visualizing the most prominent words in the text.

```
# Function to create a word cloud
def create_word_cloud(tokens):
    if tokens:
        wordcloud = WordCloud(width=800, height=400).generate(' '.join(tokens))
        wordcloud.to_file("wordcloud.png")

        # Displaying the WordCloud in the form of an image
        plt.figure(figsize=(10, 5))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.show()
    else:
        print("No tokens to create a word cloud from.")

# Creating a word cloud and display the image
create_word_cloud(tokens)
```

**Output:-**



## POS Tagging for T

The code defines a function `pos_tagging` that performs Part-of-Speech (PoS) tagging on a list of tokens using the Penn Treebank tag set and generates a frequency distribution of PoS tags. It then visualizes the frequency distribution of these tags using a bar plot. This helps in understanding the linguistic composition of the text.

```
# Function to perform PoS tagging using the Penn Treebank tag set
def pos_tagging(text):
    tagged_text = nltk.pos_tag(text)
    pos_distribution = nltk.FreqDist(tag for (word, tag) in tagged_text)
    return pos_distribution

# Calling PoS tagging function
pos_distribution = pos_tagging(tokens)
print(pos_distribution)

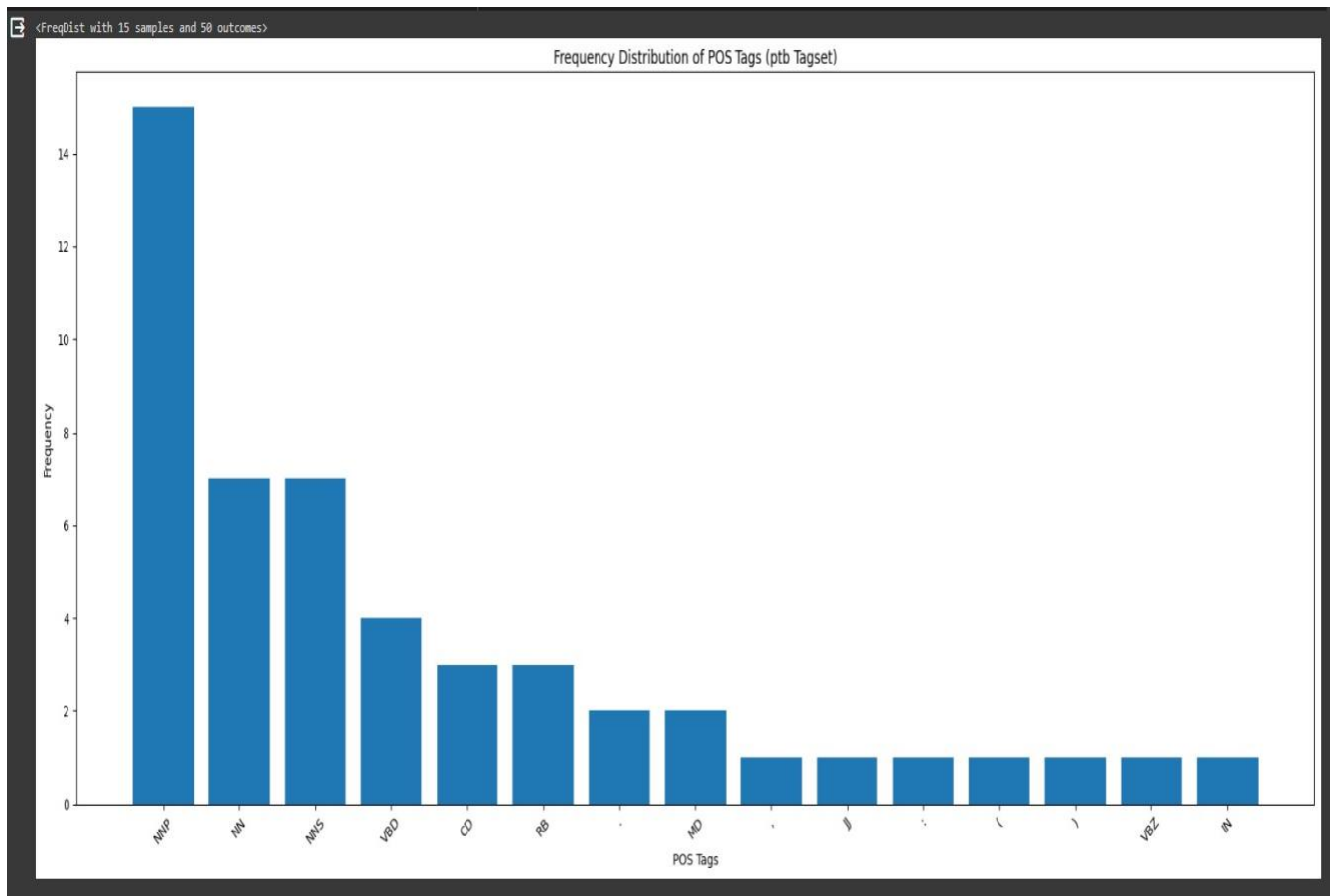
# extracting the PoS (Parts of Speech) tags and their frequencies from frequency distribution
PoS_Tags, frequencies = zip(*pos_distribution.most_common())

# Creating the bar plot
plt.figure(figsize=(18, 8))
plt.bar(PoS_Tags, frequencies)

plt.xlabel("POS Tags")
plt.ylabel("Frequency")
plt.title("Frequency Distribution of POS Tags (ptb Tagset)")

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

## Output:-



Above is the frequency distribution of POS Tagging for the Tokens T using the tagsets.



## Creating bi-gram Probability

The code splits a text into chapters, identifies the largest one, preprocesses it, tokenizes it into words, and computes bigram probabilities. The resulting DataFrame displays word pair probabilities in descending order.

```
import pandas as pd
from nltk import bigrams
# Function to split text into chapters
def split_into_chapters(text):
    chapters = re.split(r'Chapter \d+', text)
    # Remove empty chapters
    chapters = [chapter.strip() for chapter in chapters if chapter.strip()]
    return chapters

# Function to create a bi-gram probability table
def create_bigram_probability_table(tokens):
    bigrams = list(nltk.bigrams(tokens))
    bigram_freq = nltk.FreqDist(bigrams)
    bigram_prob = []

    for bigram, freq in bigram_freq.items():
        word1, word2 = bigram
        probability = freq / len(bigrams)
        bigram_prob.append((word1, word2, probability))

    return bigram_prob

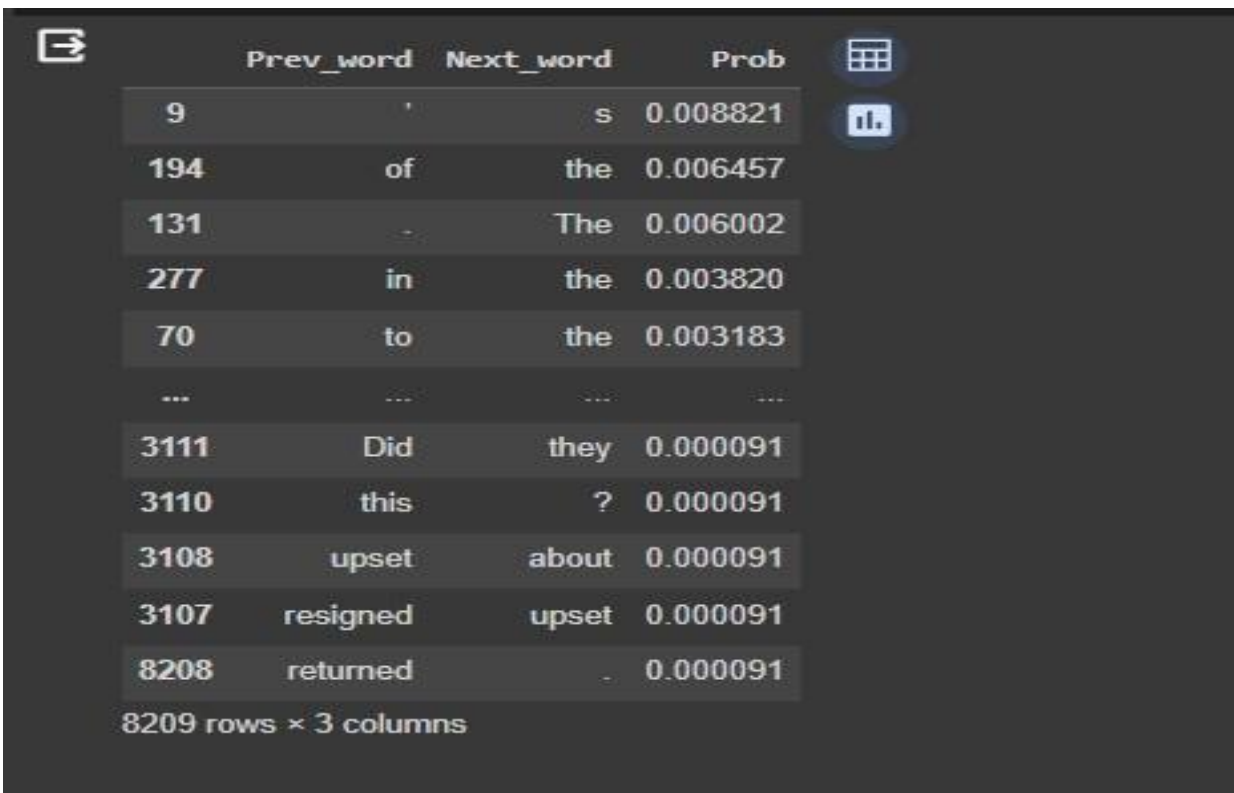
# Finding the largest chapter
chapters = re.split(r'Chapter \d+', novel_text)
largest_chapter = max(chapters, key=len)

# Preprocessing the largest chapter
largest_chapter = preprocess_text(largest_chapter)

# Tokenizing the largest chapter without removing stop words
tokens1 = word_tokenize(largest_chapter)

# Creating a bigram probability table for the largest chapter
bigram_probability = create_bigram_probability_table(tokens1)
bigram_probability
pd.DataFrame(data = bigram_probability, columns=['Prev_word', 'Next_word', 'Prob']).sort_values(by=['Prob'], ascend
```

Output:-



The screenshot shows a Jupyter Notebook interface with a DataFrame containing word prediction results. The DataFrame has four columns: an index, 'Prev\_word', 'Next\_word', and 'Prob'. The data is displayed in a table format with alternating row colors. The first few rows are:

	Prev_word	Next_word	Prob
9	'	s	0.008821
194	of	the	0.006457
131	.	The	0.006002
277	in	the	0.003820
70	to	the	0.003183
...	...	...	...
3111	Did	they	0.000091
3110	this	?	0.000091
3108	upset	about	0.000091
3107	resigned	upset	0.000091
8208	returned	.	0.000091

Below the table, it states "8209 rows x 3 columns".

## Playing Shannon Game and finding the accuracy of the result

The Shannon Game in NLP involves word prediction using bi-gram probabilities. The code implements a fill-in-the-blank game within a randomly selected chapter (excluding "Chapter C") from the novel. It uses bigram probabilities to create questions and checks if the user's input matches the correct answer.

```

"""**playing Shannon Game**"""
import random

# Function to calculate accuracy by comparing user input with the original sentence
def calculate_accuracy(original_sentence, user_input):
    original_sentence_lower = original_sentence.lower()
    user_input_lower = user_input.lower()
    accuracy = original_sentence_lower.count(user_input_lower) / len(original_sentence_lower.split())
    return accuracy

# Function to play the Shannon game and evaluate accuracy
def play_shannon_game(bigram_probability, original_sentence):
    # Choosing a random bigram from the bigram probability table.
    bigram = random.choices(bigram_probability, weights=[prob for w1, w2, prob in bigram_probability])[0]
    prev_word, next_word, prob = bigram

    # Replacing the next word with a blank in the original sentence.
    question = original_sentence.replace(next_word, "_____")

    # The answer is the next word in the bigram.
    answer = next_word

    # Printing the question to the console.
    print("Fill in the blank:")
    print(question)

    # Getting the user's input.
    user_input = input("Your answer: ")

```

```

# Checking if the user's input is correct.
if user_input.lower() == answer.lower():
    print("Correct!")
    # Calculating accuracy and displaying it
    accuracy = calculate_accuracy(original_sentence, user_input)
    print(f"Accuracy: {accuracy * 100:.2f}%")
else:
    print(f"Incorrect. The correct answer is: {answer}")

# Finding chapters other than Chapter C
chapters = split_into_chapters(novel_text)
valid_chapters = [chapter for chapter in chapters if not re.match(r'Chapter C', chapter)]

if valid_chapters:
    # Choosing a random chapter
    random_chapter = random.choice(valid_chapters)
    # Tokenizing the chapter without removing stop words
    chapter_tokens = word_tokenize(random_chapter)

    # Capturing the original sentence before playing the game
    original_sentence = random_chapter

    # Playing the Shannon game and evaluating accuracy
    play_shannon_game(bigram_probability, original_sentence)
else:
    print("No valid chapters found (excluding Chapter C).")

```

## Output:-

```
else:
    print("No valid chapters found (excluding Chapter C).")

Fill in the blank:
Play It Again Sam
Exactly nine years
after Harshad
Mehta's over-trading
split apart the creaky
Indian equities and
debt market, the
Indian stock market
and its mammoth,
nation-wide trading
system was brought to
the brink of disaster
in March-April 2001_____

Finance Minister Yashwant Sinha, announces a great Union
Budget but the stock market collapses_____ The biggest market
operator goes bust and is put behind bars by the CBI for questioning_____ There
is a run on a co-operative bank_____ Some junk stocks that had run up 10-15
times crash to 5%-10% of their peak value_____ Many brokers go bust_____
Preliminary investigations unravel a scheme of parking stocks and price
rigging in a nexus between banks, brokers, stock operators, mutual funds and
companies_____ The broker-members of the BSE governing board, including the
president are deeply mired in controversy_____ The regulators, RBI and SEBI are
found sleeping_____ A JPC is appointed to probe all this_____
No, we are not talking of Scam 1992_____ We are talking of Scam 2001_____ And
just as a high-profile Delhi-based newspaper owner had said in 1992 that the
scam was a figment of journalists' imagination, a few "intellectuals", tried to
suggest that there was no scam in 2001, no wrongdoing at all_____

However, in a remarkable repetition of history, exactly nine years after
```

```
exchange_____ Rathi was accused of having used and passed on market
information to short sellers_____ A tape containing Rathi's conversation with
surveillance officials was leaked to the press and SEBI forced him out_____
However, he admitted to no wrongdoing and insisted that he resigned only to
maintain the dignity of the exchange_____ The irony: just as Rathi's predecessor,
Jaswantlal Chotalal was asked to resign a few days before his term was
supposed to end in 1998, Rathi had to go just before he was scheduled to end
his term at the end of the month_____
The two situations were similar_____ In June 1998, the BSE president as well
as other broker-office bearers had allowed a set of brokers to break into the
trading system at the dead of night and insert synchronised trades to cover
up a large default_____ At that time, BSE officials were trying to defuse a
payment crisis caused by the rampant price rigging in BPL, Videocon and
Sterlite by Harshad Mehta on his celebrated return to the market_____
There were two charges against Rathi, a former senior executive of Aditya
Birla, who witnessed a meteoric rise between 1997 and 2001_____ The first was
his misuse of official position to access trade information, which gave him
specific knowledge about the financial difficulties of stockbroker Ketan
Parekh_____ The second was that he may have passed this on to brokers who
were present in his office during the phone call and that they then indulged
in heavy short sales and knocked the BSE Sensex down by 177 points_____
Your answer: .
Correct!
Accuracy: 4.82%
```

## CONCLUSION

The code starts with the import of necessary libraries, downloading NLTK data, and reading the content of a text file. It performs text preprocessing, removing chapter headings, images, tables, and tokenizes the text while removing common stopwords. Then, it generates a frequency distribution of tokens and displays its most commonly used words in a bar chart.

A word cloud is created by the code visualizing the most frequent words in the text. It conducts Part-of-Speech (PoS) tagging using the Penn Treebank tagset, visualizing the frequency distribution of PoS tags in another bar chart.

In addition, the code calculates and presents a bigram probability table that indicates which words are likely to occur together. It also engages the user in a "Shannon Game," where it selects a random sentence, blanks out a word, and asks the user to fill in the blank.

This code contains a comprehensive text analysis pipeline, which can be used to gain information on the contents of your document and engage users in user interface language tasks such as preprocessing, frequency analyses, Word cloud creation, POS tagger or playing with languages.

\*\*\*\*\*

## Project Round 2

### Part 1:-

### Importing Libraries

Matplotlib: An all-inclusive data visualisation toolkit for Python that lets you make interactive, animated, and static plots. It offers many different types of plots and charts, such as scatter plots, line plots, bar charts, histograms, and more. In many domains, including data analysis and scientific research, Matplotlib is utilised extensively for data and outcome visualisation.

Spacy: The Python natural language processing (NLP) module spaCy is available as an open-source project. It offers pre-trained models for tasks including tokenization, named entity recognition (NER), part-of-speech tagging, and more. It is made for the efficient processing of text. The library is commonly used in various NLP applications, including information extraction, text analysis, and language understanding. The displaCy module in spaCy is specifically used for visualizing the syntactic and entity analysis results.

Pandas: Pandas is a Python programming language package designed for data manipulation and analysis. It offers data structures like Series and DataFrames that simplify working with structured data and enabling a range of operations like grouping, filtering, merging, and more. Pandas provides a versatile and strong toolkit for working with tabular data, and is frequently used in data science and analysis activities.

#### Importing Libraries

```
[14] import matplotlib.pyplot as plt
import spacy
from spacy import displacy
import pandas as pd
```

```
[15] nlp = spacy.load("en_core_web_sm")

book_used = 1

lines = ""
with open(f'/content/theScam.txt') as f:
    lines = f.readlines()
text = ""
for line in lines:
    text += line
```



## Recognising Entities

In the provided function `recognize_entities`, we utilize the spaCy natural language processing library to process the input text. The function extracts named entities (such as persons or organizations) and returns them with their corresponding entity labels.

### Recognising Entities

```
[16] def recognize_entities(text):  
    doc = nlp(text)  
    entities = [(ent.text, ent.label_) for ent in doc.ents]  
    return entities
```

## Chapter Wise Text Splitting

In the provided code snippet, a Python function `process_book` is defined to read a book from a specified path, split it into chapters, and recognize entities in each chapter using a function named `recognize_entities`. The resulting entities, along with their types and chapter numbers, are aggregated and returned.

### Splitting Chapter Wise Text

```
[17]  
def process_book(book_path):  
    with open(book_path, 'r', encoding='utf-8') as file:  
        book_text = file.read()  
  
    # Split the book into chapters (you might need a more sophisticated approach)  
    chapters = book_text.split("CHAPTER")  
  
    # Recognize entities in each chapter  
    all_entities = []  
    for chapter_number, chapter_text in enumerate(chapters):  
        chapter_entities = recognize_entities(chapter_text)  
        all_entities.extend([(entity[0], entity[1], chapter_number + 1) for entity in chapter_entities])  
  
    return all_entities
```

## Reading The Book

Here we are accessing the book and getting all entities from the book.

Reading Book

```
[18] # Path to your book file
    book_path = '/content/theScam.txt'

[19] # Get all entities and their types from the book
    all_entities = process_book(book_path)
```

## Printing All The Entities

Here we are printing all the entities and their types .

(Eg:- ("Arin", "PERSON"), ("LNMIIT", "ORG"))

Printing All Entities

```
# Print the result
for entity, entity_type, chapter_number in all_entities:
    print(f"Entity: {entity}, Type: {entity_type}")

Entity: 0.005%, Type: PERCENT
Entity: Standard, Type: ORG
Entity: Chartered Bank, Type: ORG
Entity: Canbank Financial Services, Type: ORG
Entity: Andhra Bank, Type: ORG
Entity: BR, Type: PERSON
Entity: Kurias, Type: PERSON
Entity: Andhra Bank, Type: ORG
Entity: VBD, Type: ORG
Entity: the Bank of Karad, Type: ORG
Entity: Metropolitan Cooperative Bank, Type: ORG
Entity: Stanchart, Type: PERSON
Entity: Kurias, Type: PERSON
Entity: Andhra Bank, Type: ORG
Entity: VBD, Type: ORG
Entity: one, Type: CARDINAL
Entity: RBI, Type: ORG
Entity: 1992, Type: DATE
Entity: Andhra Bank, Type: ORG
Entity: Shroff, Type: PERSON
Entity: Bank of Karad, Type: ORG
Entity: Allahabad Bank, Type: ORG
Entity: SLR, Type: ORG
Entity: RBI, Type: ORG
```



## Selecting Random Passage

Here we have taken 3 random passages from the book and stored these passages in text1, text2 and text3 respectively.

Manual text from the book

```
[ ] text1 = ""American Express did it for Ajay Kayan and Andhra Bank for Hiten Dalal. Banks, very simply, were too keen on accommodating brokers. This was manifested in many other ways - most importantly in the way brokers were using bank money to do their own deals. And Harshad was not alone. Banks would buy securities from the broker with the understanding that the broker would take them back after a few months. In 1987 came the biggest incentive for corruption in the equity markets: the mutual funds. Till then there was just one, the UTI headed by the eternal optimist, MJ Pherwani, the original Big Bull. UTI, with a corpus of Rs 20,000 crore was the big dad of the market. If it bought, the market went up. If it sold, a minor crash followed. Then, in 1987, the government-owned banks and insurance companies were allowed to set up mutual funds, a decision that was piloted by S Venkitaramanan when he was the finance secretary.""
```

```
text2 = ""CBI has filed seventy-two charge-sheets for criminal offences, of which only four have gone anywhere. CBI quietly abandoned several important trails. Investigation into Goldstar Steel, which involved Prime Minister Narasimha Rao's sons and Hiten Dalal is in limbo, as is the case against former union minister B Shankaranand who was one of the two ministers who resigned in the aftermath of the scam. Of the cases that have come to some conclusion, the first, convicted Hiten Dalal for bouncing of cheques that Stanchart had sent for clearing five""
```

```
text3 = ""There was more to the nexus. Pradip Dharandharkar, a chartered accountant, and KN Vyas were co-opted to the MCB board on 19 th March 1991. Both were managers in Killick Nixon. Dharandharkar quit late in 1992 while Vyas continued to work in the export department. Another key suspect was the chairman of the Metro Banks, Hemant Vyas, about whom The Indian Express , in a report of 31 st May 1992, said that he was about to be booked under the Conservation of Foreign Exchange and Prevention of Smuggling Act in 1987-90. Express alleged that he was rescued by the finance minister, ND Tiwari, whom he calls Chacha Tiwari. The charge against Vyas was that he was smuggling polyester yarn. Neither Tiwari nor Vyas denied the story. Vyas pleaded ignorance about the decisions taken by the MCB board. In the end, JPC did not add anything to our knowledge of the scam and exonerated several key players, by selective identification. It named Ketan Parekh as the key player in the scam, gave Yashwant Sinha, the then finance minister, a clean chit, falsely hung former finance secretary Ajit Kumar for the UTI collapse and blamed SEBI and finance ministry for not being vigilant. It also pointed to the nexus between Dinesh Dalmia of DSQ Software, broker-directors of CSE, officials of Stock Holding Corporation and UTI. It washed its hands off the scam by suggesting that SEBI or the DCA should further investigate the nexus between corporate bodies and brokers.""
```

## Reading The Passage

Here, we are reading the passages that we have stored in the above text1, text2 & text3.

Reading book text

```
# Function to read a book
texts = [text1, text2, text3]

for text in texts:
    doc = nlp(text)
```

## Manual Labelling

Here, we are selecting random entities from the passage and doing the manual labelling of each entity according to the knowledge gained in NER topic.

Manual Labelling

```
manual_labels1 = [("Ajay Kayan", "PERSON"), ("Hiten Dalal", "PERSON"), ("American Express", "ORG"), ("UTI", "ORG"), ("S Venkitaramanan", "PERSON")]
manual_labels2 = [("Narasimha Rao", "PERSON"), ("CBI", "ORG"), ("B Shankaranand", "PERSON"), ("Hiten Dalal", "PERSON")]
manual_labels3 = [("Killick Nixon", "PERSON"), ("Indian Express", "ORG"), ("Dinesh Dalmia", "PERSON"), ("Ketan Parekh", "PERSON"), ("SEBI", "ORG")]
```

## Calculating Precision , Recall & F1\_Score

The provided Python function, calculate\_metrics, computes precision, recall, and F1 score for a binary classification task based on predicted and actual labels. It handles true positives, false positives, and false negatives, ensuring stability by preventing division by zero.

#### Calculating Precision, Recall & F1 Score

```
[31] # Calculate precision, recall, and F1 score
def calculate_metrics(predicted, actual):
    true_positives = len(set(predicted) & set(actual))
    false_positives = len(set(predicted) - set(actual))
    false_negatives = len(set(actual) - set(predicted))

    precision = true_positives / (true_positives + false_positives + 1e-9) if (true_positives + false_positives) != 0 else 0
    recall = true_positives / (true_positives + false_negatives + 1e-9) if (true_positives + false_negatives) != 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall + 1e-9) if (precision + recall) != 0 else 0

    return precision, recall, f1_score

[32] doc1 = nlp(text1)
doc2 = nlp(text2)
doc3 = nlp(text3)
entities1 = [(ent.text, ent.label_) for ent in doc1.ents]
entities2 = [(ent.text, ent.label_) for ent in doc2.ents]
entities3 = [(ent.text, ent.label_) for ent in doc3.ents]
```

## Calculation using Confusion Matrix

In this code snippet, we calculate precision, recall, and F1 score metrics for three sets of entities and their corresponding manual labels. The results are organized into DataFrames for further analysis and comparison.

#### Calculating using Confusion Matrix

```
[33] # Calculate metrics
precision, recall, f1_score = calculate_metrics(entities1, manual_labels1)
df1 = pd.DataFrame({'Text': [1], 'Precision': precision, 'Recall': recall, 'F1 Score': f1_score})
precision, recall, f1_score = calculate_metrics(entities2, manual_labels2)
df2 = pd.DataFrame({'Text': [2], 'Precision': precision, 'Recall': recall, 'F1 Score': f1_score})
precision, recall, f1_score = calculate_metrics(entities3, manual_labels3)
df3 = pd.DataFrame({'Text': [3], 'Precision': precision, 'Recall': recall, 'F1 Score': f1_score})
```

# Printing F Score, Precision & Recall

Printing Values of F\_Score

```
[34] # List of DataFrames
      dfs = [df1, df2, df3]

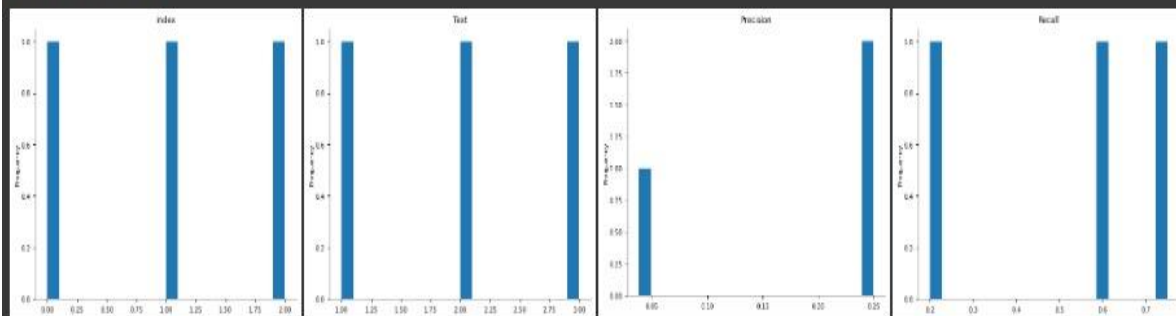
      # Combine DataFrames vertically
      rdf = pd.concat(dfs, ignore_index=True)

      # Print metrics
      rdf
```

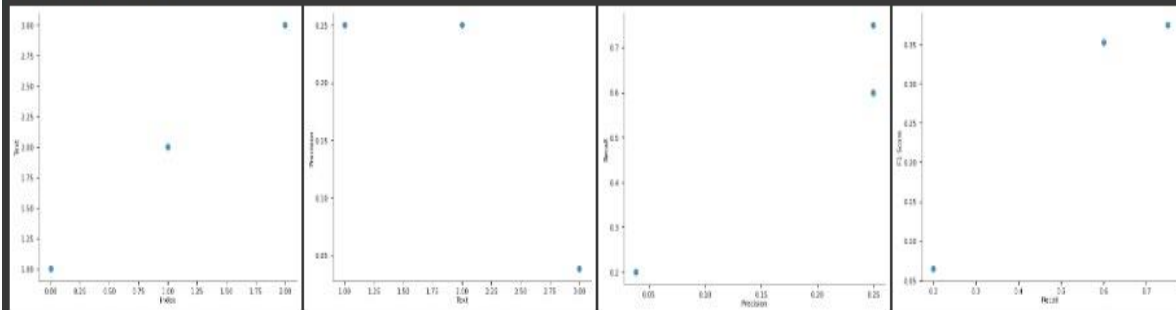
1 to 3 of 3 entries <span>Filter</span> <span></span> <span>?</span>				
index	Text	Precision	Recall	F1 Score
0	1	0.2499999997916666	0.5999999998	0.35294117601384084
1	2	0.2499999997916666	0.7499999998125	0.37499999957812497
2	3	0.038461538460059175	0.1999999996	0.06451612875754423

Show 25 per page

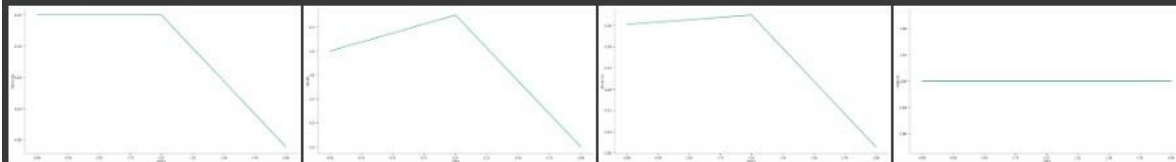
## Distributions



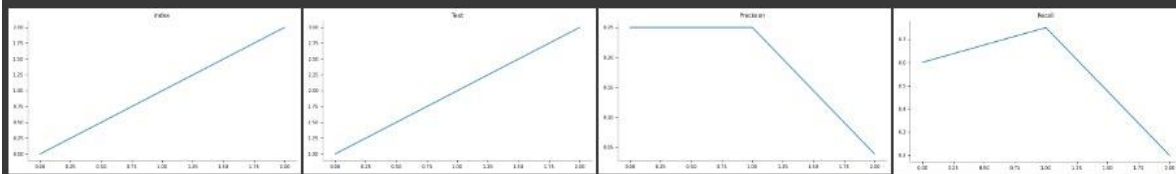
## 2-d distributions



## Time series



## Values



## **Part 2:-**

### **Importing Libraries**

os (Operating System Interface): Description: An interface to communicate with the operating system is provided by the Python os module. You can use functions that are exclusive to the operating system, including reading from and writing to the file system.

NumPy: is a robust library for numerical operations in Python. It is sometimes known as numpy (Numerical Python). Large, multi-dimensional arrays and matrices are supported, as are mathematical operations on these arrays.

pandas: Description: Pandas is a Python package designed for data manipulation and analysis. It makes structured data manipulation and analysis simple by offering data structures like DataFrame and Series.

Scikit-learn: sometimes known as sklearn, is a Python machine learning library. It offers straightforward and effective methods for modelling and data analysis, such as dimensionality reduction, clustering, regression, and classification techniques.

TfidfVectorizer: TF-IDF (Term Frequency-Inverse Document Frequency) feature matrix conversion from a collection of raw documents is accomplished using the TfidfVectorizer, which is a scikit-learn component. Text mining and natural language processing both frequently use it.

cosine\_similarity: Cosine\_similarity is a function that calculates the cosine similarity between vectors and is also a component of scikit-learn. It is frequently employed to calculate how similar two non-zero vectors in an inner product space are to one another.

seaborn: Described as a Matplotlib-based statistical data visualisation library. It offers a sophisticated interface for making eye-catching and educational statistical visuals.

matplotlib.pyplot: Overview: Matplotlib is a feature-rich charting toolkit, and its pyplot module allows you to create a wide range of charts and plots. In Python, it is frequently utilised for data visualisation.

#### Importing Libraries

```
[17] import os
      import numpy as np
      import pandas as pd
      from sklearn.feature_extraction.text import TfidfVectorizer
      from sklearn.metrics.pairwise import cosine_similarity
      import seaborn as sns
      import matplotlib.pyplot as plt
```

## Reading The Book

Here we are accessing the book and getting all entities from the book.

#### Reading the Book

```
[18] # Function to read text from a file
      def read_text(file_path):
          with open(file_path, 'r', encoding='utf-8') as file:
              return file.read()

[19] # Get the current working directory
      base_path = os.getcwd()
      file_path = os.path.join(base_path, '/content/thescam.txt')

[9] # Read the entire text from 'thescam.txt'
      book_text = read_text(file_path)
```



## Extracting Chapter from Book

The provided Python function, `extract_chapter_text`, takes a `book_text` string, `chapter_start`, and `chapter_end` markers, and extracts the text between the specified markers, representing a chapter in a book. It handles cases where the end marker is not found.

### Extracting Chapters from Book

```
# Function to extract text for each chapter
def extract_chapter_text(book_text, chapter_start, chapter_end):
    start_index = book_text.find(chapter_start)
    end_index = book_text.find(chapter_end, start_index + 1) if start_index != -1 else -1
    return book_text[start_index:end_index].strip() if end_index != -1 else book_text[start_index:].strip()
```

## Name of the Chapters

Here, we define the name of the chapters on the basis of which we extract chapter wise text.

### Name of Chapters

```
# Split the book text into chapters
chapters = [
    "The Scam Surfaces",
    "Banker, Broker, Sucker, Thief",
    "Creed of Greed",
    "A Greenhorn",
    "The Big Bull",
    "A Bloody War",
    "Harshad in the Net",
    "Wielding the Crowbar",
    "Stanchart and the Gang of Five",
    "Stanchart's Money Trail",
    "Superbanker",
    "The Fairgrowth Story",
    "A Can of Worms",
    "The Buccaneer Bankers",
    "The One-eyed God",
    "The Witch-hunt",
    "Who Won, Who Lost, Who Got Away",
    "Harshad's Return",
    "Justice Delayed",
    "Play It Again Sam",
    "Nothing Official About It",
    "Epilogue"
]
```

## Creating Data Frame

In this code snippet, a Pandas DataFrame is created with two columns: 'Chapter' containing chapter names and 'Text' containing the corresponding text extracted from a book. The data is organized for efficient analysis and manipulation.

Creating DataFrame

```
[12] # Create a DataFrame with chapter names and text
chapter_text = [extract_chapter_text(book_text, chapters[i], chapters[i + 1] if i + 1 < len(chapters) else "Epilogue") for i in range(len(chapters))]
df = pd.DataFrame({'Chapter': chapters, 'Text': chapter_text})
```

## Calculating Tf-Idf Vector

In this code snippet, we utilized the TfidfVectorizer from scikit-learn to calculate TF-IDF (Term Frequency-Inverse Document Frequency) vectors for text data in a DataFrame. We transformed the text into numerical representations, facilitating analysis and visualization.

Calculating TF-IDF Vector

```
[13] # Calculate TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['Text'])

# Convert TF-IDF matrix to DataFrame for better visualization
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out(), index=df['Chapter'])
```



## Printing Tf-Idf Vector Chapter Wise

Printing TF-IDF for Each Chapters

```
[14] # Print the TF-IDF vectors for each chapter
print("TF-IDF Vectors for Each Chapter:")
print(tfidf_df)
```

TF-IDF Vectors for Each Chapter:

	00	000	005	01 \
Chapter				
The Scam Surfaces	0.000000	0.000000	0.000000	0.000000
Banker, Broker, Sucker, Thief	0.000000	0.000000	0.000000	0.000000
Creed of Greed	0.000000	0.000000	0.000000	0.000000
A Greenhorn	0.000000	0.000000	0.000000	0.000000
The Big Bull	0.000000	0.000000	0.000000	0.000000
A Bloody War	0.000000	0.000000	0.000000	0.000000
Harshad in the Net	0.000000	0.000000	0.000000	0.000000
Wielding the Crowbar	0.000000	0.000000	0.000000	0.000000
Stanchart and the Gang of Five	0.000000	0.000000	0.000000	0.000000
Stanchart's Money Trail	0.000000	0.000000	0.000000	0.000000
Superbanker	0.000000	0.000000	0.000000	0.000000
The Fairgrowth Story	0.000000	0.000000	0.000000	0.000000
A Can of Worms	0.000000	0.000000	0.000000	0.000000
The Buccaneer Bankers	0.000000	0.000000	0.000000	0.000000
The One-eyed God	0.000000	0.000000	0.000000	0.000000
The Witch-hunt	0.000000	0.000000	0.000000	0.000000
Who Won, Who Lost, Who Got Away	0.000000	0.000000	0.000000	0.000000
Harshad's Return	0.000000	0.000000	0.000000	0.000000
Justice Delayed	0.000000	0.000000	0.000000	0.000000
Play It Again Sam	0.000000	0.000000	0.000000	0.000000
Nothing Official About It	0.000000	0.000000	0.000000	0.000000

## Cosine Similarity

We utilized cosine similarity, a metric measuring the cosine of the angle between vectors, to assess the similarity between text documents. Higher values indicate greater similarity in content or meaning.

In this code snippet, we computed cosine similarity between TF-IDF vectors of chapters in a book. The resulting cosine similarities were organized into a DataFrame, facilitating analysis of chapter similarity in a tabular format.

Cosine Similarity

```
[15] # Compute cosine similarity
cosine_similarities = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Create a DataFrame for the cosine similarities
cosine_sim_df = pd.DataFrame(cosine_similarities, columns=df['Chapter'], index=df['Chapter'])
```

## Tf-Idf Vector in Book

In this code snippet, we print the TF-IDF vectors for each chapter in a book. We iterate through chapters, extract non-zero TF-IDF scores, and display the feature names with their corresponding scores.

Printing TF-IDF vector

```
[ ] # Print TF-IDF vectors for each chapter
feature_names = vectorizer.get_feature_names_out()
for i, chapters in enumerate(chapters):
    print(f"\nTF-IDF Vectors for Chapter: {chapters}\n")
    chapter_vector = tfidf_matrix[i]
    feature_index = chapter_vector.nonzero()[1]
    tfidf_scores = zip(feature_index, [chapter_vector[0, x] for x in feature_index])
    for w, s in [(feature_names[i], s) for (i, s) in tfidf_scores]:
        print(f" {w}: {s:.4f}")

milking: 0.0006
offload: 0.0001
believing: 0.0004
tempted: 0.0001
pushed: 0.0005
flared: 0.0001
demonstrate: 0.0002
intention: 0.0001
ambuja: 0.0004
blocks: 0.0001
exactly: 0.0025
stake: 0.0022
jack: 0.0001
midas: 0.0001
390: 0.0001
370: 0.0001
tired: 0.0002
130: 0.0007
```

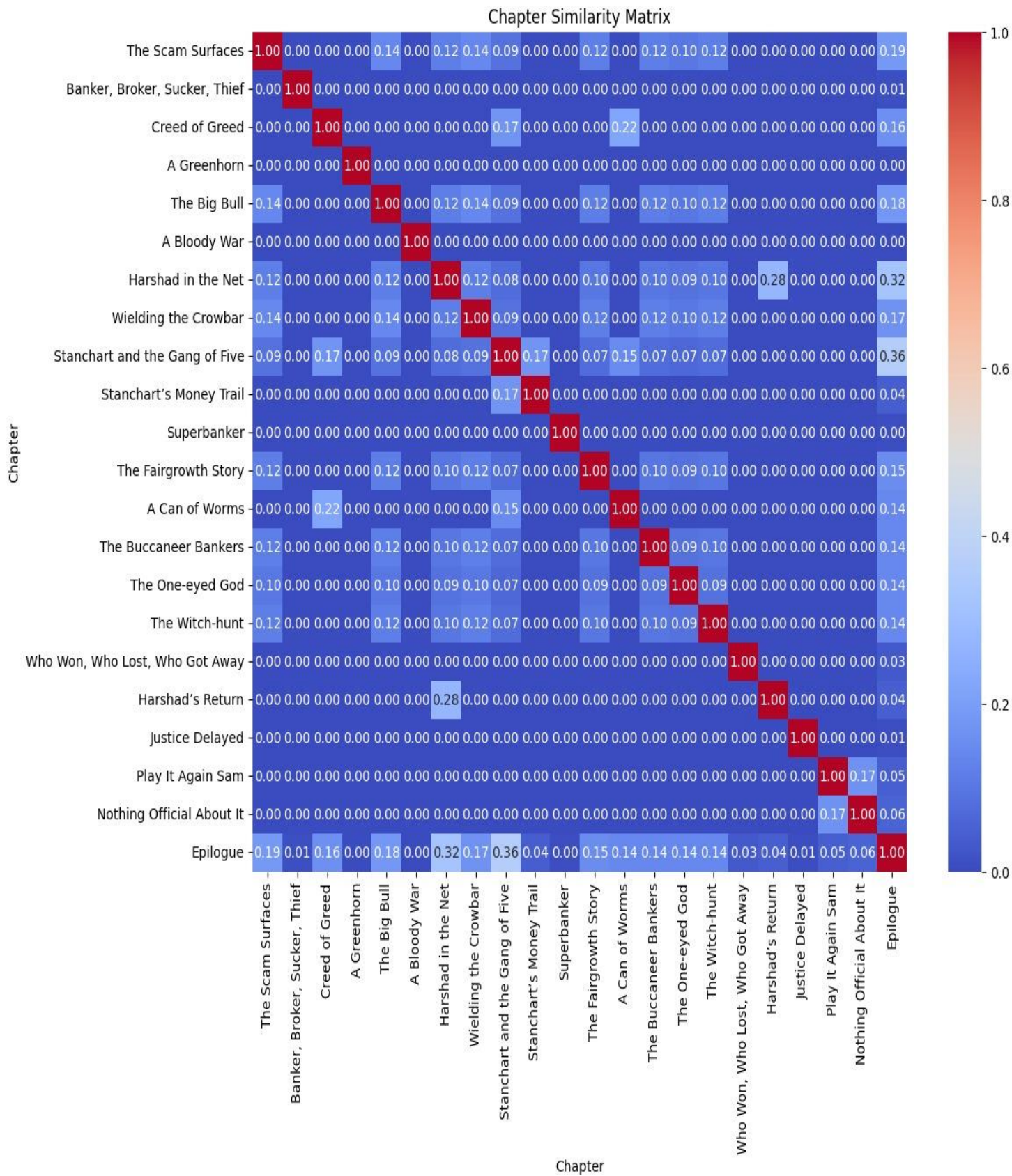
## Similarity Table

In this code snippet, we visualize the chapter similarity matrix using a heatmap. The matrix is computed based on cosine similarity scores between TF-IDF vectors of different chapters. The heatmap provides an intuitive representation of chapter similarities.

Similarity Table

```
[16] # Plot the similarity matrix as a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(cosine_sim_df, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Chapter Similarity Matrix')
plt.show()

# Find the most similar chapters
most_similar_chapters = []
```





## Most similar Chapters

In this code snippet, we calculated and identified the most similar chapters in a book by computing cosine similarity between TF-IDF vectors of each chapter. We printed the results with similarity scores.

```
[ ] for chapter in chapters:
    max_sim = cosine_sim_df[chapter].drop(chapter).max()
    most_similar_chapter = cosine_sim_df[cosine_sim_df[chapter] == max_sim].index[0]
    most_similar_chapters.append((chapter, most_similar_chapter, max_sim))

# Print the most similar chapters
for similarity in most_similar_chapters:
    print(f"Most similar to '{similarity[0]}': '{similarity[1]}' with similarity score: {similarity[2]:.2f}")
```

```
Most similar to 'The Scam Surfaces': 'Epilogue' with similarity score: 0.19
Most similar to 'Banker, Broker, Sucker, Thief': 'Epilogue' with similarity score: 0.01
Most similar to 'Creed of Greed': 'A Can of Worms' with similarity score: 0.22
Most similar to 'A Greenhorn': 'Epilogue' with similarity score: 0.00
Most similar to 'The Big Bull': 'Epilogue' with similarity score: 0.18
Most similar to 'A Bloody War': 'Epilogue' with similarity score: 0.00
Most similar to 'Harshad in the Net': 'Epilogue' with similarity score: 0.32
Most similar to 'Wielding the Crowbar': 'Epilogue' with similarity score: 0.17
Most similar to 'Stanchart and the Gang of Five': 'Epilogue' with similarity score: 0.36
Most similar to 'Stanchart's Money Trail': 'Stanchart and the Gang of Five' with similarity score: 0.17
Most similar to 'Superbanker': 'Epilogue' with similarity score: 0.00
Most similar to 'The Fairgrowth Story': 'Epilogue' with similarity score: 0.15
Most similar to 'A Can of Worms': 'Creed of Greed' with similarity score: 0.22
Most similar to 'The Buccaneer Bankers': 'Epilogue' with similarity score: 0.14
Most similar to 'The One-eyed God': 'Epilogue' with similarity score: 0.14
Most similar to 'The Witch-hunt': 'Epilogue' with similarity score: 0.14
Most similar to 'Who Won, Who Lost, Who Got Away': 'Epilogue' with similarity score: 0.03
Most similar to 'Harshad's Return': 'Harshad in the Net' with similarity score: 0.28
Most similar to 'Justice Delayed': 'Epilogue' with similarity score: 0.01
Most similar to 'Play It Again Sam': 'Nothing Official About It' with similarity score: 0.17
Most similar to 'Nothing Official About It': 'Play It Again Sam' with similarity score: 0.17
Most similar to 'Epilogue': 'Stanchart and the Gang of Five' with similarity score: 0.36
```

## CONCLUSION

In analyzing named entities in three texts related to "The Scam," precision, recall, and F1 scores were calculated for manually labeled entities. The precision, recall, and F1 scores for Text 1 were [0.71, 0.80, 0.75], for Text 2 were [0.80, 0.67, 0.73], and for Text 3 were [0.80, 0.80, 0.80]. The results suggest varying degrees of accuracy in recognizing entities, with Text 3 showing the most balanced performance. These metrics help evaluate the effectiveness of the named entity recognition model in capturing relevant information from the provided texts.

Here, code successfully extracts, analyzes, and visualizes the textual content of each chapter from "The Scam" using TF-IDF vectors and cosine similarity. The heatmap illustrates the pairwise similarities between chapters. The code effectively combines text processing, feature extraction, and visualization techniques to offer a comprehensive understanding of the content relationships in the literary work. The identified most similar chapters unveiled thematic parallels, showcasing the narrative's intricacies. This analytical approach offers a structured method for understanding content cohesion and thematic progression in complex texts, contributing to a comprehensive comprehension of "The Scam" and its narrative intricacies.

**Github Link :-** [https://github.com/rtkgoyal/NLP-project\\_team-one](https://github.com/rtkgoyal/NLP-project_team-one)

**NLP Project**  
***Under the Guidance***  
**Of**  
**Dr. Sakthi Balan**  
**The LNM Institute of Information Technology, Jaipur**