GENERATIVE AI LAB MANUAL

VI Semester AI&DS

Designed By,

1. **Prof. Vaibhav Chavan**
2. **Dr. Aijaz Qazi**

Institute Vision

To become premier institute committed to **academic excellence** and **global competence** for the **holistic development** of students.

Key words: academic excellence, global competence, holistic development.

Institute Mission

M1: Develop competent human resources, adopt outcome based education (OBE) and implement cognitive assessment of students.

M2: Inculcate the traits of global competencies amongst the students.

M3: Nurture and train our students to have domain knowledge, develop the qualities of global professionals and to have social consciousness for holistic development.

Department Vision

To deliver a **quality and responsive** education in the field of artificial intelligence and data science emphasizing **professional skills** to face **global challenges** in the evolving IT paradigm.

Key words: quality and responsive, professional skills, global challenges.

Department Mission

M1: Leverage multiple pedagogical approaches to impart knowledge on the current and emerging AI technologies.

M2: Develop an inclusive and holistic ambiance that bolsters problem solving, cognitive abilities and critical thinking.

M3: Enable students to develop trust worthiness, team spirit, understanding law-of-the-land, social behavior to be a global stake holder.

Program Specific Outcomes (PSOs):

PSO1: To apply core knowledge of Artificial Intelligence, Machine Learning, Deep Learning, Data Science, Big Data Analytics and Statistical Learning to develop effective solutions for real-world problems.

PSO2: To demonstrate proficiency in specialized and emerging technologies such as Natural Language Processing, Cloud Computing, Robotic Process Automation, Storage Area Networks and the Internet of Things to meet the stringent and diverse professional challenges.

PSO3: To imbibe managerial skills, social responsibility, ethical and moral values through courses in Management and Entrepreneurship, Software Engineering Principles, Universal Human Values and Ability Enhancement Programs to meet the industry and societal expectations.

Program Educational Objectives (PEOs)

PEO 1: Build a strong foundation in mathematics, core programming, artificial intelligence, machine learning, and data science to enable graduates to analyze, design, and implement intelligent systems for solving complex real-world problems.

PEO 2: Foster creativity, cognitive and research skills to analyze the requirements and technical specifications of software to articulate novel engineering solutions for an efficient product design.

PEO 3: Prepare graduates for dynamic career opportunities in AI and Data Science by equipping them with interdisciplinary knowledge, adaptability, and practical exposure to tools and techniques required for industry and research.

PEO 4: Instill a strong sense of ethics, professional responsibility, and human values, empowering graduates to contribute positively to society and lead with integrity in their professional domains.

PEO 5: Encourage graduates to pursue higher education, certification program, entrepreneurial ventures, etc. by nurturing a mindset of continuous learning and awareness of global trends and challenges.

Experiment 1

Aim: Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.

Code:

```
import gensim.downloader as api
def load():
    print("Loading word vectors.")
    model = api.load("word2vec-google-news-300")
    print("Loaded!")
    return model
def vec_arith(model):
    print("\nExploring word relationships using word vectors")
    result = model.most_similar(positive=["GOOD", "BAD"], negative=["HAPPY"], topn=1)
    print("GOOD + BAD - HAPPY =", result)

    result = model.most_similar(positive=["INDIA", "CANADA"], negative=["DELHI"], topn=1)
    print("INDIA + CANADA - DELHI =", result)

def main():
    model = load()
    vec_arith(model)

if __name__ == "__main__":
    main()
```

Output:

```
Loading word vectors.
Loaded!

Exploring word relationships using word vectors
GOOD + BAD - HAPPY = [('TERRIBLE', 0.45916804671287537)]
INDIA + CANADA - DELHI = [('NEW_ZEALAND', 0.5123980641365051)]
```

Experiment 2

Aim: Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.

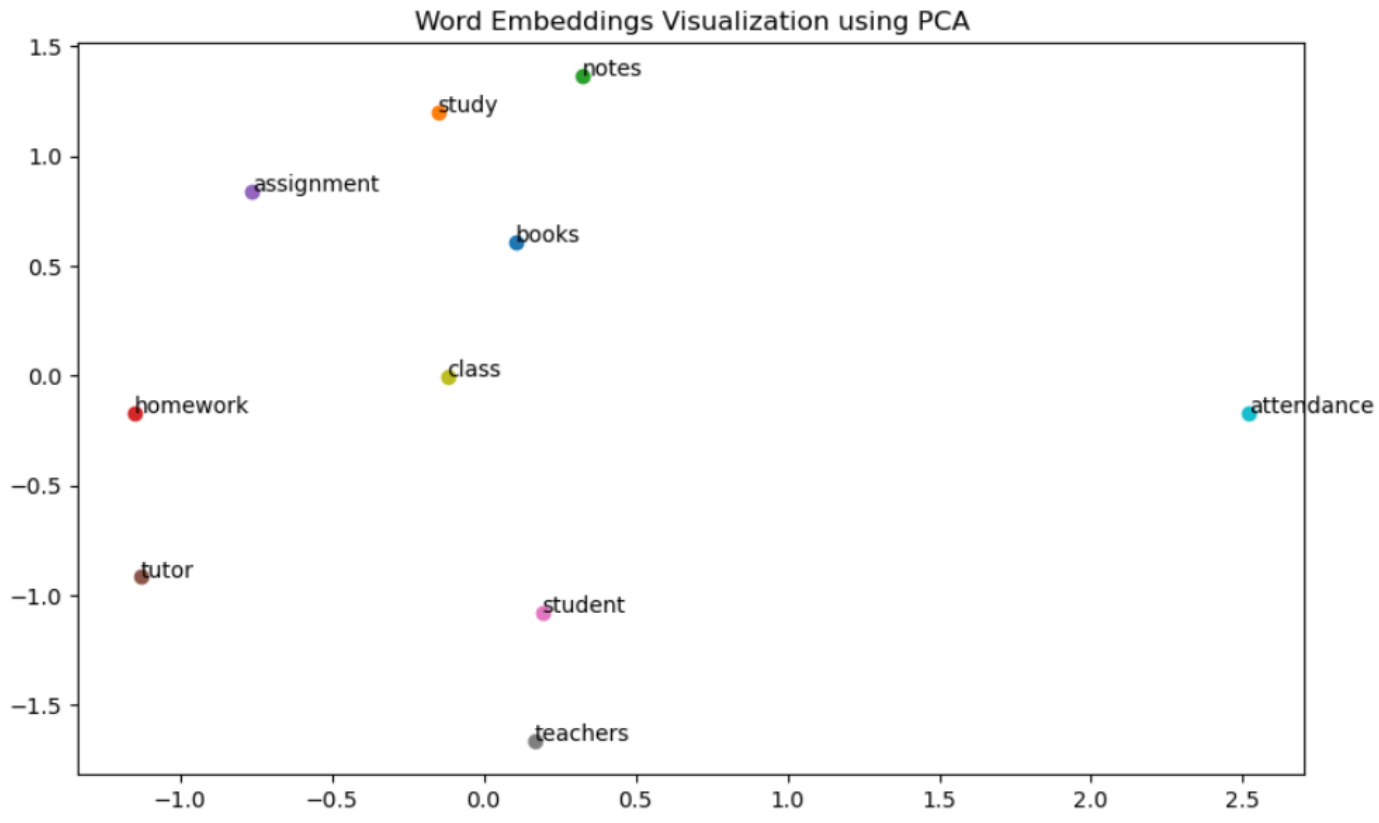
Code:

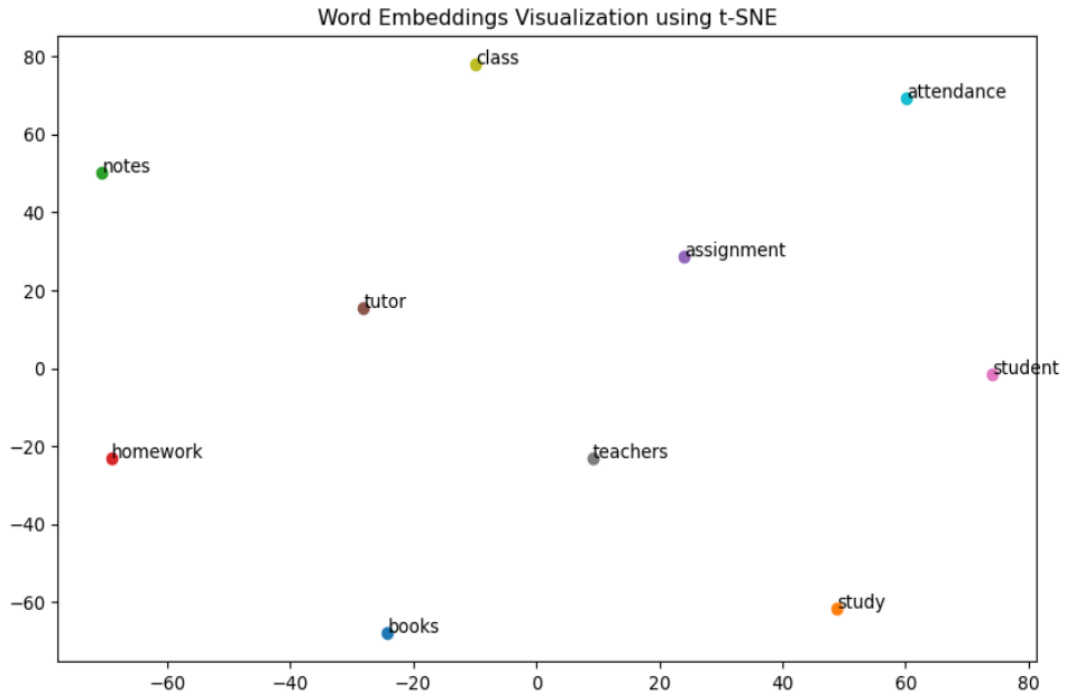
```
import gensim.downloader as api
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import numpy as np
def load():
    print("Loading pre-trained word vectors...")
    model = api.load("word2vec-google-news-300")
    print("Model loaded successfully!")
    return model
def vector_arithmetic(model):
    print("\nExploring word relationships using vector arithmetic:")
    examples = [
        (["king", "woman"], ["man"]),
        (["Paris", "Italy"], ["France"]),
        (["walking", "run"], ["walk"])
    ]
    for pos, neg in examples:
        try:
            result = model.most_similar(positive=pos, negative=neg, topn=1)
            print(f"{pos} - {neg} = {result}")
        except KeyError as e:
            print(f"Word '{e.args[0]}' not found in vocabulary.")
def visualize_embeddings(model, words, method='PCA'):
    valid_words = [word for word in words if word in model.key_to_index]
    vectors = np.array([model[word] for word in valid_words])

    if method == 'PCA':
        reducer = PCA(n_components=2)
    else:
        reducer = TSNE(n_components=2, random_state=42, perplexity=min(30, len(valid_words)-1))
    reduced_vectors = reducer.fit_transform(vectors)
    plt.figure(figsize=(10, 6))
    for i, word in enumerate(valid_words):
        plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
        plt.annotate(word, (reduced_vectors[i, 0], reduced_vectors[i, 1]))
```

```
    plt.title(f"Word Embeddings Visualization using {method}")
    plt.show()
def find_similar_words(model, word, topn=5):
    if word in model.key_to_index:
        return model.most_similar(word, topn=topn)
    else:
        return f"Word '{word}' not in vocabulary"
def main():
    model = load()
    tech_words = ["books", "study", "notes", "homework", "assignment", "tutor", "student", "teachers",
"class", "attendance"]
    visualize_embeddings(model, tech_words, method='PCA')
    visualize_embeddings(model, tech_words, method='t-SNE')
    word = "education"
    similar_words = find_similar_words(model, word)
    print(f"Top 5 words similar to '{word}':", similar_words)
if __name__ == "__main__":
    main()
```

Output:

```
Loading pre-trained word vectors...
Model loaded successfully!
```






```
Top 5 words similar to 'education': [('eduction', 0.7980327010154724), ('eduation', 0.7175651788711548), ('LISA_MICHALS_covers', 0.6817381381988525), ('M
att_Krupnick_covers', 0.6798164248466492), ('educational', 0.6780007481575012)]
```

Experiment 3

Aim: Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal, medical) and analyze how embeddings capture domain-specific semantics.

Code:

Step 1:

```
import nltk
nltk.download('punkt')
```

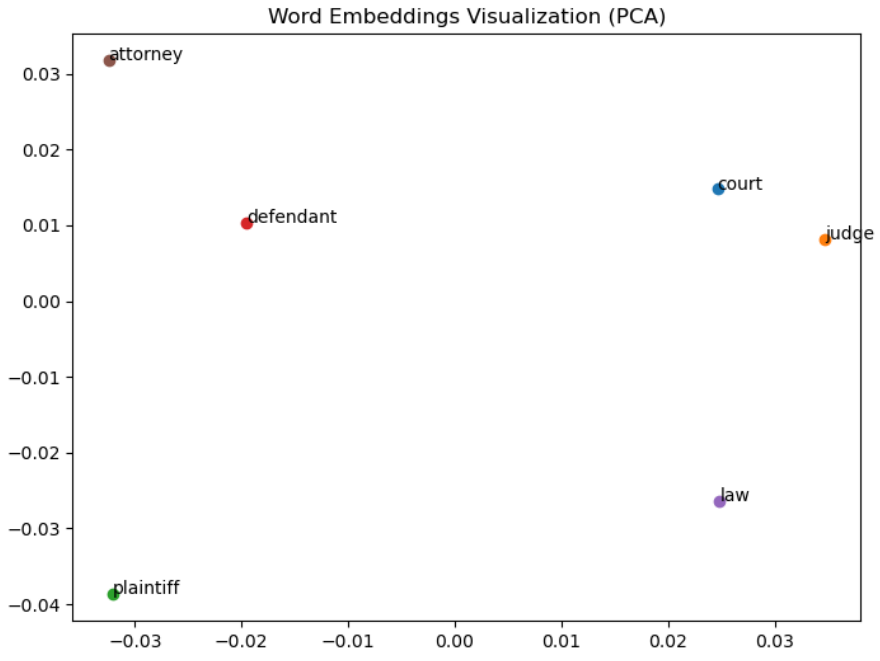
Step 2:

```
legal_corpus = [
    "The court ruled in favor of the defendant in the civil case.",
    "Intellectual property law protects patents, copyrights, and trademarks.",
    "The plaintiff filed a lawsuit against the corporation for breach of contract.",
    "A judge must ensure due process is followed in all criminal trials.",
    "Legal precedents set by the Supreme Court influence lower court decisions.",
    "The attorney argued that the evidence was inadmissible in court."
]
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
import string
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
def preprocess_text(corpus):
    cleaned_corpus = []
    for sentence in corpus:
        tokens = word_tokenize(sentence.lower())  # Tokenize and lowercase
        tokens = [word for word in tokens if word.isalnum()]  # Remove punctuation
        cleaned_corpus.append(tokens)
    return cleaned_corpus
tokenized_corpus = preprocess_text(legal_corpus)
print("Tokenized Corpus:", tokenized_corpus)
model = Word2Vec(sentences=tokenized_corpus, vector_size=100, window=5, min_count=1, workers=4)
model.save("legal_word2vec.model")
print("Word2Vec model trained and saved successfully!")
model = Word2Vec.load("legal_word2vec.model")
word = "judge"
if word in model.wv:
    print(f"\nWords similar to '{word}':")
    similar_words = model.wv.most_similar(word, topn=5)
```

```
    for w, sim in similar_words:
        print(f"{w}: {sim:.4f}")
else:
    print(f"'{word}' not found in vocabulary.")
words = ["court", "judge", "plaintiff", "defendant", "law", "attorney"]
word_vectors = np.array([model.wv[word] for word in words if word in model.wv])
valid_words = [word for word in words if word in model.wv]
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)
plt.figure(figsize=(8, 6))
for i, word in enumerate(valid_words):
    plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
    plt.annotate(word, (reduced_vectors[i, 0], reduced_vectors[i, 1]))
plt.title("Word Embeddings Visualization (PCA)")
plt.show()
```

Output:

```
Tokenized Corpus: [['the', 'court', 'ruled', 'in', 'favor', 'of', 'the', 'defendant', 'in', 'the', 'civil', 'case'], ['intellectual', 'property', 'law',
'protects', 'patents', 'copyrights', 'and', 'trademarks'], ['the', 'plaintiff', 'filed', 'a', 'lawsuit', 'against', 'the', 'corporation', 'for', 'breac
h', 'of', 'contract'], ['a', 'judge', 'must', 'ensure', 'due', 'process', 'is', 'followed', 'in', 'all', 'criminal', 'trials'], ['legal', 'precedents',
'set', 'by', 'the', 'supreme', 'court', 'influence', 'lower', 'court', 'decisions'], ['the', 'attorney', 'argued', 'that', 'the', 'evidence', 'was', 'in
admissible', 'in', 'court']]
Word2Vec model trained and saved successfully!

Words similar to 'judge':
lawsuit: 0.2855
due: 0.2705
trials: 0.2599
court: 0.1887
argued: 0.1424
```



Experiment 4

Aim: Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using word embeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for the original and enriched prompts. Compare the outputs in terms of detail and relevance.

Code:

```
import gensim.downloader as api
from gensim.models import KeyedVectors
import numpy as np
print("Loading word embeddings...")
word_vectors = api.load("glove-wiki-gigaword-50")
print("Word embeddings loaded.")
def get_similar_words(word, topn=5):
    try:
        similar_words = word_vectors.most_similar(word, topn=topn)
        return [word for word, similarity in similar_words]
    except KeyError:
        return []
def enrich_prompt(original_prompt):
    words = original_prompt.split()
    enriched_words = []
    for word in words:
        similar_words = get_similar_words(word, topn=2)  # Get 2 similar words per word
        if similar_words:
            enriched_words.append(f"{word} ({', '.join(similar_words)})")
        else:
            enriched_words.append(word)
    return " ".join(enriched_words)
def mock_genai_response(prompt):
    if len(prompt.split()) > 10:  # Enriched prompts are longer
        return f"Generated response to '{prompt}': This is a detailed and nuanced answer with rich context and
additional insights."
    else:
        return f"Generated response to '{prompt}': This is a concise and straightforward answer."
original_prompt = "Students from my class are tired of my subject"
enriched_prompt = enrich_prompt(original_prompt)
print(f"Original Prompt: {original_prompt}")
print(f"Enriched Prompt: {enriched_prompt}")
original_response = mock_genai_response(original_prompt)
enriched_response = mock_genai_response(enriched_prompt)
print("\nResponses:")
print(f"Original Response: {original_response}")
print(f"Enriched Response: {enriched_response}")
```

```
def compare_outputs(original, enriched):
    original_length = len(original.split())
    enriched_length = len(enriched.split())
    print("\nComparison:")
    print(f"Original Response Length: {original_length} words")
    print(f"Enriched Response Length: {enriched_length} words")
    print("Detail & Relevance Analysis:")
    if enriched_length > original_length:
        print("The enriched prompt produced a more detailed response due to added context from similar words.")
    else:
        print("The enriched prompt did not significantly improve detail.")
compare_outputs(original_response, enriched_response)
```

Output:

```
Loading word embeddings...
Word embeddings loaded.
Original Prompt: Students from my class are tired of my subject
Enriched Prompt: Students from (in, while) my (your, me) class (classes, type) are (other, these) tired (scared, feel) of (which, in) my (your, me) subject (legal, terms)

Responses:
Original Response: Generated response to 'Students from my class are tired of my subject': This is a concise and straightforward answer.
Enriched Response: Generated response to 'Students from (in, while) my (your, me) class (classes, type) are (other, these) tired (scared, feel) of (which, in) my (your, me) subject (legal, terms)': This is a detailed and nuanced answer with rich context and additional insights.

Comparison:
Original Response Length: 19 words
Enriched Response Length: 41 words
Detail & Relevance Analysis:
The enriched prompt produced a more detailed response due to added context from similar words.
```

Experiment 5

Aim: Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word. Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generates similar words. Constructs a short paragraph using these words.

Code:

```
import random
import gensim.downloader as api
def load_model():
    return api.load("glove-wiki-gigaword-100")
def get_similar_words(model, seed, topn=5):
    if seed in model:
        return [w for w, _ in model.most_similar(seed, topn=topn)]
    else:
        return []
def make_paragraph(seed, similar_words):
    words = [seed] + similar_words
    random.shuffle(words)
    sentence = " ".join(f"{w}" for w in words[:-1])
    sentence += f" and finally {words[-1]}"
    return (
        f"Once upon a time, the notion of “{seed}” sparked imagination. "
        f"As ideas took shape, characters and scenes burst forth: {sentence}. "
        "Thus began a tale unlike any other."
    )
def main():
    seed = input("Enter a seed word: ").strip().lower()
    print("Loading model…")
    model = load_model()
    print(f"Finding words similar to '{seed}'…")
    similars = get_similar_words(model, seed, topn=5)
    if not similars:
        print(f"Sorry, '{seed}' is not in the model vocabulary.")
        return
    print("Similar words:", similars)
    paragraph = make_paragraph(seed, similars)
    print("\nGenerated Paragraph:\n")
    print(paragraph)
if __name__ == "__main__":
    main()
```

Output:

```
Enter a seed word:  cricket
Loading model…
Finding words similar to 'cricket'…
Similar words: ['rugby', 'twenty20', 'england', 'indies', 'cricketers']

Generated Paragraph:

Once upon a time, the notion of “cricket” sparked imagination. As ideas took shape, characters and scenes burst forth: twenty20 cricketers england rugby
indies and finally cricket. Thus began a tale unlike any other.
```

Experiment 6

Aim: Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load the sentiment analysis pipeline. Analyze the sentiment by giving sentences to input.

Code:

Step 1:

```
!pip install transformers
```

Step 2:

```
from transformers import pipeline
sentiment_analyzer = pipeline("sentiment-analysis")
sentences = [
    "I love using Hugging Face models, they are amazing!",
    "The weather today is terrible and I feel so sad.",
    "This is the best day of my life!",
    "I'm not sure how I feel about this."
]
results = sentiment_analyzer(sentences)
for sentence, result in zip(sentences, results):
    print(f"Sentence: {sentence}")
    print(f"Sentiment: {result['label']}, Confidence: {result['score']:.4f}")
    print("-" * 50)
```

Output:

```
Sentence: I love using Hugging Face models, they are amazing!
Sentiment: POSITIVE, Confidence: 0.9999
--------------------------------------------------
Sentence: The weather today is terrible and I feel so sad.
Sentiment: NEGATIVE, Confidence: 0.9992
--------------------------------------------------
Sentence: This is the best day of my life!
Sentiment: POSITIVE, Confidence: 0.9999
--------------------------------------------------
Sentence: I'm not sure how I feel about this.
Sentiment: NEGATIVE, Confidence: 0.9992
--------------------------------------------------
```

Experiment 7

Aim: Summarize long texts using a pre-trained summarization model using Hugging face model. Load the summarization pipeline. Take a passage as input and obtain the summarized text.

Code:

```
from transformers import pipeline

summarizer = pipeline("summarization")
passage = """
India, country that occupies the greater part of South Asia. It is made up of
28 states and eight union territories, and its national capital is New Delhi,
built in the 20th century just south of the historic hub of Old Delhi to serve
as India’s administrative center. Its government is a constitutional republic
that represents a highly diverse population consisting of thousands of ethnic
groups and hundreds of languages. India became the world’s most populous country
in 2023, according to estimates by the United Nations.
"""

summary = summarizer(passage, max_length=100, min_length=25, do_sample=False)
print(summary[0]['summary_text'])
```

Output:

```
 It is made up of 28 states and eight union territories, and its national capital is New Delhi . India became the world’s most populous country in 2023,
according to estimates by the United Nations .
```

Experiment 8

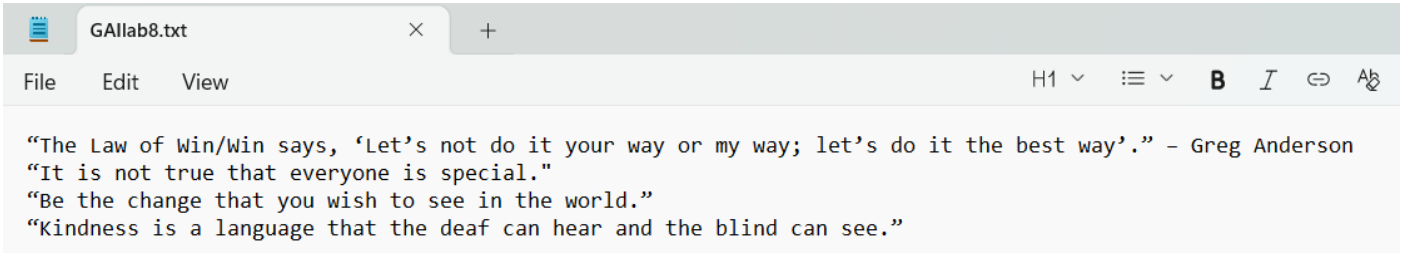
Aim: Install langchain, cohere (for key), langchain-community. Get the api key(By logging into Cohere and obtaining the cohere key). Load a text document from your google drive . Create a prompt template to display the output in a particular manner.

Code:

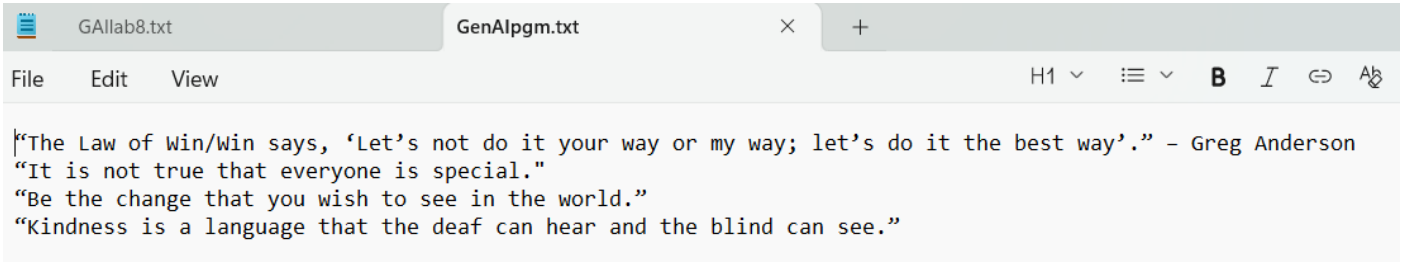
```
1)!pip install langchain
2)!pip install cohere
3)!pip install langchain-community
4)!pip install google-auth
5)!pip install google-auth-oauthlib
6)!pip install google-auth-httplib2
7)!pip install google-api-python-client==2.126.0

import os
os.environ["COHERE_API_KEY"] = "YOUR_COHERE_API_KEY"

import os
import io
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload
SCOPES = ['https://www.googleapis.com/auth/drive.readonly']
def authenticate_google_drive():
    creds = None
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file('client_secret_1076587815690-
25iejnti0a9q29fr9d1tt497efgohaar.apps.googleusercontent.com.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
    return creds
def download_file_from_drive(file_id, output_file):
    creds = authenticate_google_drive()
    service = build('drive', 'v3', credentials=creds)
    request = service.files().get_media(fileId=file_id)
```

```
    fh = io.FileIO(output_file, 'wb')
    downloader = MediaIoBaseDownload(fh, request)
    done = False
    while done is False:
        status, done = downloader.next_chunk()
        print(f"Download {int(status.progress() * 100)}%.")
    print(f"File downloaded to {output_file}")
file_id = '1r1gDsmkgX0TLj47Ke9bgDUKtzwm3uI4Q'
output_file = 'GenAIpgm.txt'
download_file_from_drive(file_id, output_file)
```

Output:




```
Download 100%.
File downloaded to GenAIpgm.txt
```



Experiment 9

Aim: Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a custom output parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia: The founder of the Institution. When it was founded. The current branches in the institution . How many employees are working in it. A brief 4-line summary of the institution.

Code:

Step 1:

```
!pip install wikipedia-api
```

Step 2:

```
from pydantic import BaseModel
from typing import List
import wikipediaapi
class InstitutionDetails(BaseModel):
    founder: str
    founded_year: int
    branches: List[str]
    employee_count: int
    summary: str
def fetch_wikipedia_data(institution_name: str) -> str:
    wiki_wiki = wikipediaapi.Wikipedia(language='en', user_agent='MyWikipediaBot/1.0')
    page = wiki_wiki.page(institution_name)
    if not page.exists():
        raise ValueError(f"Wikipedia page for '{institution_name}' not found.")
    return page.text
import re
def extract_institution_details(text: str) -> InstitutionDetails:
    founder_match = re.search(r"founded by ([\w\s]+)", text, re.IGNORECASE)
    founder = founder_match.group(1) if founder_match else "Unknown"
    year_match = re.search(r"founded in (\d{4})", text, re.IGNORECASE)
    founded_year = int(year_match.group(1)) if year_match else 0
    branches_match = re.findall(r"branches in ([\w\s]+)", text, re.IGNORECASE)
    branches = branches_match if branches_match else ["Unknown"]
    employee_match = re.search(r"(\d+,?\d*) employees", text, re.IGNORECASE)
    employee_count = int(employee_match.group(1).replace(",", "")) if employee_match else 0
    summary = "\n".join(text.split("\n")[:4])
    return InstitutionDetails(
        founder=founder,
        founded_year=founded_year,
        branches=branches,
        employee_count=employee_count,
```

```
        summary=summary
    )
def get_institution_details(institution_name: str) -> InstitutionDetails:
    wiki_text = fetch_wikipedia_data(institution_name)
    details = extract_institution_details(wiki_text)
    return details
if __name__ == "__main__":
    institution_name = "MIT"
    details = get_institution_details(institution_name)
    print(details)
```

Output:

```
founder='MIT alumni were a country' founded_year=1985 branches=['Unknown'] employee_count=0 summary='The Massachusetts Institute of Technology (MIT) is
a private research university in Cambridge, Massachusetts, United States. Established in 1861, MIT has played a significant role in the development of m
any areas of modern technology and science.\nIn response to the increasing industrialization of the United States, William Barton Rogers organized a sch
ool in Boston to create "useful knowledge." Initially funded by a federal land grant, the institute adopted a polytechnic model that stressed laboratory
instruction in applied science and engineering. MIT moved from Boston to Cambridge in 1916 and grew rapidly through collaboration with private industry,
military branches, and new federal basic research agencies, the formation of which was influenced by MIT faculty like Vannevar Bush. In the late twentie
th century, MIT became a leading center for research in computer science, digital technology, artificial intelligence and big science initiatives like t
he Human Genome Project. Engineering remains its largest school, though MIT has also built programs in basic science, social sciences, business manageme
nt, and humanities.\nThe institute has an urban campus that extends more than a mile (1.6 km) along the Charles River. The campus is known for academic
buildings interconnected by corridors and many significant modernist buildings. MIT\'s off-campus operations include the MIT Lincoln Laboratory and the
Haystack Observatory, as well as affiliated laboratories such as the Broad and Whitehead Institutes. Campus life is often noted for demanding workloads,
a hands-on approach to research and coursework, and elaborate practical jokes known as "hacks".\nAs of October 2024, 105 Nobel laureates, 26 Turing Awar
d winners, and 8 Fields Medalists have been affiliated with MIT as alumni, faculty members, or researchers. In addition, 58 National Medal of Science re
cipients, 29 National Medals of Technology and Innovation recipients, 50 MacArthur Fellows, 83 Marshall Scholars, 41 astronauts, 16 Chief Scientists of
the US Air Force, and 8 foreign heads of state have been affiliated with MIT. The institute also has a strong entrepreneurial culture and MIT alumni hav
e founded or co-founded many notable companies.'
```

Experiment 10

Aim: Build a chatbot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document, and then we'll create a chatbot that can interact with it. Users will be able to ask questions about the Indian Penal Code and have a conversation with it.

Code:

```
import json

# Load IPC JSON data
def load_ipc_data(file_path=r"D:\ipcc_multi_chapter.json"):
    with open(file_path, "r", encoding="utf-8") as file:
        return json.load(file)

# Retrieve section details
def get_section_details(ipc_data, section_number):
    for chapter, details in ipc_data["Indian Penal Code"].items():
        if "sections" in details and section_number in details["sections"]:
            return details["sections"][section_number]
    return None

# Simple chatbot function
def ipc_chatbot():
    ipc_data = load_ipc_data()
    print("\n⚖️ Welcome to the IPC Chatbot! Ask me about any IPC section (e.g., 'Tell me about Section 6'). Type 'exit' to quit. ⚖️\n")

    while True:
        user_input = input("You: ").strip().lower()

        if user_input == "exit":
            print("Exiting IPC Chatbot. Have a great day!")
            break

        # Extract section number
        words = user_input.split()
        section_number = None
        for i, word in enumerate(words):
            if word == "section" and i + 1 < len(words):
                section_number = "Section " + words[i + 1]
                break

        if section_number:
```

```
            details = get_section_details(ipc_data, section_number)
            if details:
                print(f"\n{section_number}: {details['title']}\n{details['description']}\n")
            else:
                print(f" Sorry, I couldn't find {section_number} in the IPC data.")
        else:
            print(" Please ask about a specific section (e.g., 'Tell me about Section 6').")

# Run the chatbot
if __name__ == "__main__":
    ipc_chatbot()

# In[ ]:
```

Output:

```
⚖️ Welcome to the IPC Chatbot! Ask me about any IPC section (e.g., 'Tell me about Section 6'). Type 'exit' to quit. ⚖️

You:  Tell me about Section 300

Section 300: Murder
Defines murder.

You:  Tell me about Section 302

Section 302: Punishment for Murder
Punishment for committing murder.

You:  Tell me about Section 307

Section 307: Attempt to Murder
Deals with attempt to commit murder.

You:  Tell me about Section 359

Section 359: Kidnapping
Defines kidnapping.

You:  Tell me about Section 499

Section 499: Defamation
Defines defamation.

You:  Tell me about Section 390

Section 390: Robbery
Defines robbery.

You:  exit
Exiting IPC Chatbot. Have a great day!
```