

```

# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
sugataghosh_google_word2vec_path = kagglehub.dataset_download('sugataghosh/google-word2vec')
sanyatargrenkin_cc_en_300_bin_path = kagglehub.dataset_download('sanyatargrenkin/cc-en-300-bin')

print('Data source import complete.')


# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session


#pgm1

!pip install --upgrade gensim scipy


#pgm2
#PCA or TSNE

!pip install --upgrade gensim scipy


from gensim.models.fasttext import load_facebook_model
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
model_path="/kaggle/input/cc-en-300-bin/cc.en.300.bin"
fasttext_model=load_facebook_model(model_path)
print("fasttext model loaded successfully!")

```

```
king=fasttext_model.wv["king"]
queen=fasttext_model.wv["queen"]
print("king vector: ",king[:5])
print("queen vector: ",queen[:5])
```

```
tech_words=['Lawyer','Judge','Court','Fruit','Quantum','Encryption','database','computernetworks','Cybersecurity','Artificialintelligence']
for i in tech_words:
    print(i)
```

```
word_vectors=np.array([fasttext_model.wv[word] for word in tech_words])
word_vectors[0],word_vectors[1]
```

```
pca=PCA(n_components=2)
embeddings_2d=pca.fit_transform(word_vectors)
plt.figure(figsize=(10,8))
plt.scatter(embeddings_2d[:,0],embeddings_2d[:,1],marker='o',color='blue')
for i,word in enumerate(tech_words):
    plt.annotate(word,(embeddings_2d[i,0],embeddings_2d[i,1]),fontsize=12)
plt.title("2D PCA projection of technology word embeddings")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid()
plt.show
```

```
import gensim
word_vectors = gensim.models.KeyedVectors.load_word2vec_format("/kaggle/input/google-word2vec/GoogleNews-vectors-negative300.bin",binary=True)
def find_similar_words(word, top_n=5):
    try:
        similar_words = word_vectors.most_similar(word, topn=top_n)
        return similar_words
    except KeyError:
        return f"The word '{word}' is not in the vocabulary."
word = "king"
similar_words = find_similar_words(word)
print(f"Top 5 similar words to '{word}':")
for sim_word, similarity in similar_words:
    print(f"{sim_word}: {similarity}")
```

```
#pgm3
```

```
!pip install gensim nltk
```

```
import nltk
from nltk.tokenize import sent_tokenize,word_tokenize
nltk.download('punkt')
medical_corpus=[
    "The patient was diagnosed with hypertension after several visits to the clinic.",
    "In clinical trials, new medications are often tested for their efficacy and side effects.",
    "The physician recommended a follow-up appointment to monitor the patient's recovery.",
    "A balanced diet and regular exercise are essential for maintaining cardiovascular health.",
    "The nurse administered the vaccine to the patient as part of the immunization program.",
    "MRI scans are frequently used to assess the condition of the brain and spinal cord.",
```

```

    "Chronic conditions, such as diabetes, require ongoing management to prevent complications."
]

tokenized_corpus=[word_tokenize(sentence.lower()) for sentence in medical_corpus]
print(tokenized_corpus)

print("Training Word2Vec model...")
model=Word2Vec(sentences=tokenized_corpus,vector_size=100>window=5,min_count=1,workers=4,epochs=50)
print("Model Training Complete!")

words=list(model.wv.index_to_key)
embeddings=np.array([model.wv[word] for word in words])
tsne=TSNE(n_components=2,random_state=42,perplexity=5,n_iter=300)
tsne_result=tsne.fit_transform(embeddings)
plt.figure(figsize=(10,8))
plt.scatter(tsne_result[:,0],tsne_result[:,1],color="blue")
for i,word in enumerate(words):
    plt.text(tsne_result[i,0]+0.02,tsne_result[i,1]+0.02,word,fontsize=12)
plt.title("Word Embeddings Visualization")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.grid()
plt.show

def find_similar_words(input_word,top_n=5):
    try:
        similar_words=model.wv.most_similar(input_word,topn=top_n)
        print(f"Words similar to '{input_word}':")
        for word,similarity in similar_words:
            print(f"{word}({similarity:.2f})")
    except KeyError:
        print(f"'{input_word}' not found in vocabulary.")
find_similar_words("vaccine")

#pgm4

!pip install groq
!pip install gensim

import groq
from gensim.models import KeyedVectors
import numpy as np
import os

def load_word_vectors():
    model_path="/kaggle/input/google-word2vec/GoogleNews-vectors-negative300.bin"
    model=KeyedVectors.load_word2vec_format(model_path,binary=True)
    return model
model=load_word_vectors()

```

```

# Get the vector for the word "king"
king_vector = model['king']

# Display the vector
print(king_vector)

from kaggle_secrets import UserSecretsClient
def generate_response(prompt,model_name="llama-3.3-70b-versatile"):
    user_secrets=UserSecretsClient()
    groq_api_key=user_secrets.get_secret("GROQ_API_KEY")
    if not groq_api_key:
        raise ValueError("GROQ_API_KEY environment variable is not set.")
    client=groq.Client(api_key=groq_api_key)
    response=client.chat.completions.create(
        model=model_name,
        messages=[{"role":"system","content":"You are a helpful AI assistant."},
                    {"role":"user","content":prompt}]
    )
    return response.choices[0].message.content
original_prompt="What is the scope of computer science at present."
response=generate_response(original_prompt)
print(response)

def enrich_prompt(prompt,model,max_enrichments=2):
    words=prompt.split()
    enriched_words=[]
    for word in words:
        similar_words=get_similar_words(word,model,top_n=max_enrichments)
        filtered_similar_words=[w for w in similar_words if w.isalpha()]
        if filtered_similar_words:
            enriched_words.append(word + "("+",".join(filtered_similar_words)+")")
        else:
            enriched_words.append(word)
    return " ".join(enriched_words)
enriched_prompt=enrich_prompt(original_prompt,model)
print("original prompt:",original_prompt)
print("enriched prompt:",enriched_prompt)

original_response=generate_response(original_prompt)
enriched_response=generate_response(enriched_prompt)
print("\nOriginal Response:\n",original_response)
print("\nEnriched Response:\n",enriched_response)

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
def analyze_responses(original_response,enriched_response):
    vectorizer=TfidfVectorizer()
    tfidf_matrix=vectorizer.fit_transform([original_response,
                                           enriched_response])

    similarity_score=cosine_similarity(tfidf_matrix[0:1],
                                       tfidf_matrix[1:2])[0][0]

    original_length=len(original_response.split())
    enriched_length=len(enriched_response.split())
    print("\n Response Analysis:")
    print("Similarity score:",round(similarity_score,4))

```

```

print("original response word count:",original_length)
print("enriched response word count:",enriched_length)
analyze_responses(original_response,enriched_response)

#pgm5

!pip install gensim scipy nltk

from gensim.models import KeyedVectors
import numpy as np
import os
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import sent_tokenize
nltk.download('wordnet')
nltk.download('punkt')

def load_word_vectors():
    model_path='/kaggle/input/google-word2vec/GoogleNews-vectors-negative300.bin'
    model=KeyedVectors.load_word2vec_format(model_path,binary=True)
    return model
model=load_word_vectors()

def get_similar_words(word,model,top_n=5):
    try:
        similar_words=model.most_similar(word,topn=top_n)
        return [w[0] for w in similar_words]
    except KeyError:
        return []
def get_synonyms(word):
    try:
        synonyms=set()
        for syn in wordnet.synsets(word):
            for lemma in syn.lemmas():
                synonyms.add(lemma.name())
        return list(synonyms)[:5]
    except LookupError:
        return []
def generate_story(seed_word,model):
    similar_words=get_similar_words(seed_word,model,top_n=3)
    synonyms=get_synonyms(seed_word)
    word_choices=list(set(similar_words + synonyms))
    while len(word_choices)<5:
        word_choices.append(seed_word)
    story_template=(
        f"Once upon a time, in a mystical land, there was an ancient {seed_word}."
        f"Legends spoke of its power hidden within the {word_choices[0]}."
        f"One evening, under a {word_choices[1]} sky, a young explorer named Alex set out on a journey."
        f"Guided by an old {word_choices[2]}, they discovered a secret passage leading to a hidden realm."
        f"Inside, they found an inscription written in {word_choices[3]} and uncovered the secret of {word_choices[4]}."
        f"This adventure would change their fate forever, unlocking mysteries long forgotten."
    )
    return " ".join(sent_tokenize(story_template))

```

```
seed_word="adventure"
generated_story=generate_story(seed_word,model)
print("Generated Story:")
print(generated_story)
```

```
#pgm6
```

```
!pip install -q transformers
```

```
# Use a pipeline as a high-level helper
from transformers import pipeline
```

```
sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert/distilbert-base-uncased-finetuned-sst-2-english")
reviews=[
    "Great sound and super portable! Perfect for outdoor use.",
    "Decent quality, but bass could be better. Connectivity issues at times.",
    "Ideal for travel! Compact, great sound, and water-resistant.",
    "Disappointing. Muffled sound and frequent connection drops.",
    "Good sound and long battery life, but a bit bulky."
]
results=sentiment_pipeline(reviews)
for review, result in zip(reviews,results):
    print(f"Review: {review}\nsentiment: {result['label']} (confidence: {result['score']:.2f})\n")
```

```
#pgm7
```

```
!pip install transformers
```

```
from transformers import pipeline
summarizer=pipeline("summarization",model="facebook/bart-large-cnn")
long_text="""
Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans.
It involves technologies such as machine learning, natural language processing, and robotics, allowing computers to perform tasks that would typically require human cognition, such as processing data, solving problems, and making decisions. AI is rapidly transforming industries, enhancing efficiency, and creating new opportunities, but it also raises important ethical and societal questions regarding privacy, employment, and the future of work.
"""
summary=summarizer(long_text,max_length=130,min_length=30,do_sample=False)
print(summary[0]['summary_text'])
```

```
#pgm8
```

```
!pip install langchain langchain-community langchain-cohere
!pip install google-auth google-auth-oauthlib google-auth-httpplib2 google-api-python-client
!pip install cohere
```

```
import os
from kaggle_secrets import UserSecretsClient
user_secrets=UserSecretsClient()
cohere_api_key=user_secrets.get_secret("COHERE_API_KEY")
os.environ["COHERE_API_KEY"]=cohere_api_key
```

```
https://drive.google.com/file/d/1XIQ40bzxUytoYeJofRp6KTT5vyp7bEt/view?usp=sharing
```

```
!pip install gdown
!gdown --id 1XIQ40bzxUytoYeJofRp6KTT5vyp7bEt
```

```
text_content=''
with open('/kaggle/working/data.txt','r',encoding='utf-8') as file:
    text_content=file.read()
    print(text_content)
```

```
from langchain_cohere import ChatCohere
from langchain_cohere.llms import Cohere
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.documents import Document
from langchain.chains.combine_documents import(
    create_stuff_documents_chain,
)
llm=ChatCohere(
    cohere_api_key=cohere_api_key,model="command-a-03-2025"
)
print(text_content)
prompt=ChatPromptTemplate.from_messages(
    [("human","how is ai useful:\n\n {context}")]
)
chain=create_stuff_documents_chain(llm,prompt)
docs=[
    Document(page_content=text_content)
]
chain.invoke({"context":docs})
```

```
#pgm9
```

```
!pip install langchain langchain-community langchain-groq
!pip install wikipedia pydantic
!pip install groq
```

```
import os
from kaggle_secrets import UserSecretsClient
user_secrets=UserSecretsClient()
groq_api_key=user_secrets.get_secret("GROQ_API_KEY")
os.environ["GROQ_API_KEY"]=groq_api_key
```

```
from pydantic import BaseModel,Field
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain_core.prompts import ChatPromptTemplate
from langchain_groq import ChatGroq
from typing import Optional
import re
import groq
client=groq.Client(api_key=groq_api_key)
```

```

class InstitutionInfo(BaseModel):
    name:str=Field(...,description="Name of the institution")
    founder:Optional[str]=Field(None,description="Founder of the institution")
    founded_year:Optional[str]=Field(None,description="Year the institution was founded")
    branches:Optional[str]=Field(None,description="Current branches in the institution")
    employees:Optional[str]=Field(None,description="Number of employees in the institution")
    summary:Optional[str]=Field(None,description="Breif 4-line summary of the institution")

def parse_wikipedia_content(content:str)->InstitutionInfo:
    founder_match=re.search(r'(?i)founder[s]+[:\-\s]+([\^\n\r]*)',content)
    founded_match=re.search(r'(?i)founded[:\-\s]+(\d{4})',content)
    branches_match=re.search(r'(?i)campus|branches[:\-\s]+([\^\n\r]*)',content)
    employees_match=re.search(r'(?i)staff|employees[:\-\s]+([\d,]+)',content)
    summary=" ".join(content.split(".")[4:])+". "
    return InstitutionInfo(
        name="Unknown",
        founder=founder_match.group(1)if founder_match else "Not Available",
        founded_year=founded_match.group(1)if founded_match else "Not Available",
        branches=branches_match.group(1)if branches_match else "Not Available",
        employees=employees_match.group(1)if employees_match else "Not Available",
        summary=summary
    )

def fetch_institution_info(institution_name:str)->InstitutionInfo:
    try:
        page_content=wikipedia.page(institution_name).content
        institution_info=parse_wikipedia_content(page_content)
        institution_info.name=institution_name
        return institution_info
    except wikipedia.exceptions.PageError:
        return InstitutionInfo(name=institution_name,summary="No wikipedia page found.")
    except wikipedia.exceptions.DisambiguationError as e:
        return InstitutionInfo(name=institution_name,summary=f"Multiple results found:{e.options[:5]}")

llm=ChatGroq(model_name="llama-3.3-70b-versatile")
prompt=PromptTemplate(
    input_variable=["institution_name"],
    template="""
    Extract the following details about {institution_name} from wikipedia:
    -Founder
    -Founded Year
    -Current Branches
    -Number of employees
    A brief 4-line summary
    """
)
chain=LLMChain(llm=llm,prompt=prompt)
institution_name="T John Institute of Technology"
response=chain.run(institution_name=institution_name)
print(response)

```

#pgm10

```

!pip install langchain langchain-community langchain-groq
!pip install groq

```



```

!pip install PyMuPDF faiss-cpu langchain requests

import os
from kaggle_secrets import UserSecretsClient
user_secrets=UserSecretsClient()
groq_api_key=user_secrets.get_secret("GROQ_API_KEY")
os.environ["GROQ_API_KEY"]=groq_api_key

from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain_core.prompts import ChatPromptTemplate
from langchain_groq import ChatGroq
llm=ChatGroq(model_name="llama-3.3-70b-versatile",api_key=groq_api_key)
llm

import fitz
def extract_text_from_pdf(pdf_path):
    """Extracts text from a PDF file and returns it as a string."""
    doc=fitz.open(pdf_path)
    text=""
    for page in doc:
        text+=page.get_text("text")+"\n"
    return text

pdf_text=extract_text_from_pdf("/kaggle/input/ipc-document/ipc.pdf")
print("Extracted text from IPC PDF:",len(pdf_text),"characters")

import faiss
import numpy as np
from langchain.text_splitter import RecursiveCharacterTextSplitter
from sentence_transformers import SentenceTransformer

hf_model=SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
def create_faiss_index(text):
    """Chunks IPC text and stores embeddings in FAISS."""
    text_splitter=RecursiveCharacterTextSplitter(chunk_size=500,chunk_overlap=50)
    texts=text_splitter.split_text(text)
    embeddings=hf_model.encode(texts)
    d=embeddings.shape[1]
    index=faiss.IndexFlatL2(d)
    index.add(np.array(embeddings))
    return index,texts

ipc_faiss_index,ipc_chunks=create_faiss_index(pdf_text)
print("FAISS Index created with",len(ipc_chunks),"chunks.")

def retrieve_ipc_section(query):
    """Find the most relevant IPC section based on user query."""
    query_embedding=hf_model.encode([query])
    distances,indices=ipc_faiss_index.search(np.array(query_embedding),k=1)
    return ipc_chunks[indices[0][0]] if indices[0][0] < len(ipc_chunks)else "No relavant section"
query="What is the punishment for theft under IPC?"

```

```
retrieved_section=retrieve_ipc_section(query)
print("\nRelevant IPC Section:\n",retrieved_section)

prompt=PromptTemplate(
    input_variables=["ipc_section","query"],
    template="""
    You are an expert in Indian law, A user asked: "{query}"
    Based on the Indian Penal Code(IPC),the relevant section is:
    (ipc_section)

    Please provide:
    -A simple explanation
    -The Key legal points
    -Possible punishments
    -A real-world example
    """
)
def query_groq(prompt):
    response=chain.run()
    print(response)
    return response
def ipc_chatbot(query):
    related_section=retrieve_ipc_section(query)
    chain=LLMChain(llm=llm,prompt=prompt)
    response=chain.run(ipc_section=related_section,query=query)
    return response
user_query=input("Enter your legal question:")
chatbot_response=ipc_chatbot(user_query)
print(chatbot_response)
```