```
6) %pip install --upgrade --quiet huggingface_hub

%pip install --upgrade langchain

from transformers import pipeline

sentiment_analyzer = pipeline("sentiment-analysis")

sentences = [good and bad sentences]

results = sentiment_analyzer(sentences)

for sentence, result in zip(sentences, results):

   print(f"Sentence: {sentence}\nSentiment: {result['label']}, Confidence:
{result['score']:.2f}\n")
```

```
7) from transformers import pipeline

summarizer = pipeline("summarization")

long_text = """long sentences."""

summary = summarizer(long_text, max_length=100, min_length=50, do_sample=False)

print("Summarized Text:\n", summary[0]['summary_text'])

!pip install langchain-huggingface

from langchain_huggingface import HuggingFaceEndpoint

from getpass import getpass

HUGGINGFACEHUB_API_TOKEN = getpass()

import os

os.environ["HUGGINGFACEHUB_API_TOKEN"] = HUGGINGFACEHUB_API_TOKEN

text = f""" very long paragraph

"""

import requests

API_URL = "https://api-inference.huggingface.co/models/facebook/bart-large-cnn"

headers = {"Authorization": "Bearer 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"}

def query(payload):

 response = requests.post(API_URL, headers=headers, json=payload)

 return response.json()

output = query({"inputs": text})
```

```
8) !pip install langchain langchain-cohere langchain-community
!pip install gdown
import getpass
import os
if not os.environ.get("COHERE_API_KEY"):
  os.environ["COHERE_API_KEY"] = getpass.getpass("Enter API key for Cohere: ")
from langchain_cohere import ChatCohere
model = ChatCohere(model="command-r7b-12-2024")
from langchain_core.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_template("Tell me a quote on the {topic}")
chain = prompt | model
chain.invoke({"topic": "AI"}).content
import gdown
file_id = "18opmXTc4DKEPvtBKoAhNp5YUDwyJ8nA1"
file_path = "ai_agents_info.txt"
gdown.download(f"https://drive.google.com/uc?export=download&id={file_id}", file_path, quiet=False)
with open(file_path, "r", encoding="utf-8") as file:
  document_text = file.read()
print(document_text)
from langchain_core.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_template("Extract and list the types of AI agents as bullet points from the following text:{document_text}")
chain = prompt | model
print(chain.invoke({"document_text": document_text}).content)
```

```
9) from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from pydantic import BaseModel
class InstitutionDetails(BaseModel):
  """
  Pydantic model to structure the output data for institution
```

```
 details.
  """
  founder: str
  founded: str
  branches: int
  employees: int
  summary: str
prompt_template = """
Given the name of an institution, extract the following details from
Wikipedia:
1. Founder of the institution
2. When it was founded
3. Current branches of the institution
4. How many employees work in it
5. A 4-line brief summary of the institution
Institution: {institution_name}
"""
import getpass
!pip install langchain-cohere
import os
if not os.environ.get("COHERE_API_KEY"):
  os.environ["COHERE_API_KEY"] = getpass.getpass("Enter API key forCohere: ")
from langchain_cohere import ChatCohere
model = ChatCohere(model="command-r7b-12-2024")
prompt = PromptTemplate(input_variables=["institution_name"],template=prompt_template)
chain = LLMChain(llm=model, prompt=prompt)
def fetch_institution_details(institution_name: str):
  """
  Fetches institution details using the Langchain chain and GPT-3
  model.
  Args:
  institution_name (str): The name of the institution to fetch
 details for.
  Returns:
  str: The result from the LLMChain run, containing institution
  details.
  """
  result = chain.run(institution_name=institution_name)
  return result
institution_name = input("Enter the institution name: ")
institution_details = fetch_institution_details(institution_name)
print(institution_details)
```

```
5) !pip install sentence_transformers
!pip install langchain-huggingface
!pip install tf-keras --user
!pip install numpy==1.24.4 --user
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')
corpus = [ long paragrapgh]
corpus_embeddings = model.encode(corpus, convert_to_tensor=True)
corpus_embeddings
def generate_paragraph(seed_word, corpus, corpus_embeddings, model, top_n=5):
seed_sentence = f"Tell me more about {seed_word} in finance."
seed_embedding = model.encode(seed_sentence, convert_to_tensor=True)
similarities = util.pytorch_cos_sim(seed_embedding, corpus_embeddings)[0]
top_results = similarities.topk(top_n)
print('top_results:',top_results)
story = f"The topic of '{seed_word}' is crucial in the finance industry. "
for idx in top_results.indices:
similar_sentence = corpus[idx]
```

```
story += f"{similar_sentence} "
story += f"These concepts highlight the importance of {seed_word} in understanding financial markets and investment strategies."
return story
seed_word = "bonds"
story = generate_paragraph(seed_word, corpus, corpus_embeddings, model, top_n=5)
print(story)
```

4) pip install transformers –U

```
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
glove_input_file = "/content/glove.6B.100d.txt"
word2vec_output_file = "/content/glove.6B.100d.word2vec.txt"
glove2word2vec(glove_input_file, word2vec_output_file)
model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
print(model.most_similar("king"))
original_prompt = "Explain the importance of vaccinations in healthcare."
key_terms = ["vaccinations", "healthcare"]
similar_terms = []
for term in key_terms:
if term in model.key_to_index:
similar_terms.extend({word for word, _ in model.most_similar(term, topn=3)})
 if similar_terms:
enriched_prompt = f"{original_prompt} Consider aspects like: {', '.join(similar_terms)}."
else:
   enriched_prompt = original_prompt
print("Original Prompt:", original_prompt)
print("Enriched Prompt:", enriched_prompt)
import getpass
import os
GOOGLE_API_KEY= os.environ["GOOGLE_API_KEY"] = getpass.getpass("Enter your Google AI API key: ")
```

```
!pip install langchain_google_genai
from langchain_google_genai import ChatGoogleGenerativeAI
llm = ChatGoogleGenerativeAI( model="gemini-2.0-flash-exp", temperature=0, api_key=GOOGLE_API_KEY, max_tokens=256, timeout=None, max_retries=2,)
llm.invoke("Hi")
print(llm.invoke(original_prompt).content)
```

```
2) !pip install genism
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from gensim.models import KeyedVectors
model_100d = KeyedVectors.load_word2vec_format("/content/glove.6B.100d.word2vec.txt" , binary=False,limit=500000)
words = ['football', 'soccer', 'basketball', 'tennis','engineer','information', 'baseball', 'coach', 'goal', 'player', 'referee', 'team']
word_vectors = np.array([model_100d[word] for word in words])
pca = PCA(n_components=2)
pca_result = pca.fit_transform(word_vectors)
plt.figure(figsize=(10, 8))
for i, word in enumerate(words):
plt.scatter(pca_result[i, 0], pca_result[i, 1])
 plt.text(pca_result[i, 0] + 0.02, pca_result[i, 1], word, fontsize=12)
plt.title("PCA Visualization of Sports-related Word Embeddings (100d)")
plt.xlabel("PCA Dimension 1")
plt.ylabel("PCA Dimension 2")
plt.show()
def get_similar_words(word, model, topn=5):
similar_words = model.similar_by_word(word, topn=topn)
```

```
return similar_words
similar_words_football = get_similar_words('football', model_100d, topn=5)
print(f"Words similar to 'football': {similar_words_football}")
words_to_print = ['football', 'soccer']
for word in words_to_print:
if word in model_100d:
print(f"Vector embedding for '{word}':\n{model_100d[word]}\n")
else:
print(f"Word '{word}' not found in the embeddings model.")
```

```
1) !pip install genism
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
glove_input_file = "/content/glove.6B.100d.txt"
word2vec_output_file = "/content/glove.6B.100d.word2vec.txt"
glove2word2vec(glove_input_file, word2vec_output_file)
model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
print(model.most_similar("king"))
similar_to_mysore = model.similar_by_vector(model['mysore'], topn=5)
print(f"Words similar to 'mysore': {similar_to_mysore}")
result_vector_1 = model['actor'] - model['man'] + model['woman']
 result_1 = model.similar_by_vector(result_vector_1, topn=1)
print(f"'actor - man + woman' = {result_1}")
result_vector_2 = model['india'] - model['delhi'] + model['washington']
result_2 = model.similar_by_vector(result_vector_2, topn=3)
print(f"'India - Delhi + Washington' = {result_2}")
scaled_vector = model['hotel'] * 2
result_2 = model.similar_by_vector(scaled_vector, topn=3)
result_2
import numpy as np
normalized_vector = model['fish'] / np.linalg.norm(model['fish'])
```

```
result_2 = model.similar_by_vector(normalized_vector, topn=3)
result_2
average_vector = (model['king'] + model['woman'] + model['man']) / 3
result_2 = model.similar_by_vector(average_vector, topn=3)
result_2
glove_input_file = "/content/glove.6B.50d.txt"
word2vec_output_file = "/content/glove.6B.50d.word2vec.txt"
glove2word2vec(glove_input_file, word2vec_output_file )
model_50d = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
glove_input_file = "/content/glove.6B.100d.txt"
word2vec_output_file = "/content/glove.6B.100d.word2vec.txt"
glove2word2vec(glove_input_file, word2vec_output_file)
model_100d = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
word1 = "hospital"
word2 = "doctor"
similarity_50d = model_50d.similarity(word1, word2)
similarity_100d = model_100d.similarity(word1, word2)
print(f"Similarity (50d) between '{word1}' and '{word2}': {similarity_50d:.4f}")
print(f"Similarity (100d) between '{word1}' and '{word2}': {similarity_100d:.4f}")
distance_50d = model_50d.distance(word1, word2)
distance_100d = model_100d.distance(word1, word2)
print(f"Distance (50d) between '{word1}' and '{word2}': {distance_50d:.4f}")
print(f"Distance (100d) between '{word1}' and '{word2}': {distance_100d:.4f}")
```

```
3) !pip install genism
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
legal_corpus = [long para]
tokenized_corpus = [simple_preprocess(sentence) for sentence in legal_corpus]
```

```
legal_word2vec = Word2Vec( sentences=tokenized_corpus, vector_size=50, window=3, min_count=1, sg=1, epochs=100)
legal_word2vec.save("legal_word2vec.model")
word = "lawyer"
if word in legal_word2vec.wv:
print(f"Vector embedding for '{word}':\n{legal_word2vec.wv[word]}\n")
else:
print(f"Word '{word}' not found in the Word2Vec model.")
words_to_visualize = ["court", "plaintiff", "defendant", "agreement", "lawyer", "evidence", "contract", "settlement", "jury", "damages"]
word_vectors = [legal_word2vec.wv[word] for word in words_to_visualize]
word_vectors
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)
reduced_vectors
plt.figure(figsize=(10, 8))
for i, word in enumerate(words_to_visualize):
plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
plt.text(reduced_vectors[i, 0] + 0.002, reduced_vectors[i, 1], word, fontsize=12)
plt.title("PCA Visualization of Legal Word Embeddings")
plt.xlabel("PCA Dimension 1")
plt.ylabel("PCA Dimension 2")
plt.show()
similar_words = legal_word2vec.wv.most_similar("lawyer", topn=5)
print(f"Words similar to 'lawyer': {similar_words}")
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
enhanced_corpus = [ # Legal domain paragraph, # Medical domain paragraph]
tokenized_corpus = [simple_preprocess(sentence) for sentence in enhanced_corpus]
tokenized_corpus
```

```
domain_word2vec = Word2Vec( sentences=tokenized_corpus, vector_size=100, window=5,
min_count=1,sg=1, epochs=150 )
domain_word2vec.save("enhanced_domain_word2vec.model")
words_to_analyze = ["court", "plaintiff", "doctor", "patient", "guilty", "surgery"]
for word in words_to_analyze:
if word in domain_word2vec.wv:
print(f"Vector embedding for '{word}':\n{domain_word2vec.wv[word]}\n")
else:
print(f"Word '{word}' not found in the Word2Vec model.")
selected_words = ["court", "plaintiff", "defendant", "guilty", "jury", "patient", "doctor",
"hospital", "surgery", "emergency"]
word_vectors = [domain_word2vec.wv[word] for word in selected_words]
word_vectors
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)
reduced_vectors
plt.figure(figsize=(12, 8))
for i, word in enumerate(selected_words):
plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
plt.text(reduced_vectors[i, 0] + 0.002, reduced_vectors[i, 1], word, fontsize=12)
plt.title("PCA Visualization of Legal and Medical Word Embeddings")
plt.xlabel("PCA Dimension 1")
plt.ylabel("PCA Dimension 2")
plt.show()
```