PROGRAM 1

```
!pip install --upgrade gensim
from gensim.models import keyedvectors
from gensim.downloader import load
#Load a small pre-trained word2vec model
print("Loading the model, please wait...")
model=load('glove-wiki-gigaword-50'
#50 dimensional vectors trained on wikipedia
print("Model loaded successfully!")
#display vector for a word
word_vector=model['king']
print(f"\nVector for 'king':\n{word_vector}")
#Perform vector arithmetic: king-man+woman
result=model.most_similar(positive=['king','woman'],ne
gative=['man'],topn=1)
print(f"\n'king'-'man'+'woman'={result[0][0]} with
similarity score {result[0][1]:.2f}")
#Find similarity between 2 words
similarity=model.similarity('king','queen')
print(f"\nSimilarity between 'king' &
'queen':{similarity:.2f}")
#Find the odd one out
odd_one=model.doesnt_match(['breakfast','lunch','dinner
','car'])
print(f"\nOdd one out:{odd_one}")
```

PROGRAM 2

```
!pip install --upgrade gensim
import gensim.downloader as api
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
#Load pre-trained word vectors(Google News
Word2Vec)
word_vectors=api.load("word2vec-google-news-300")
#select 10 words related to 'Technology'
words=["Computer","Laptop","AI","Machine","Robot",
"Software","hardware","algorithm","network","cybersec
urity"]
vectors=np.array([word_vectors[word] for word in
words])
def plot_embeddings(vectors,words,method="PCA"):
  if method=="PCA":
  reduced=PCA(n_components=2).fit_transform(vectors)
  else:
    reduced=TSNE(n_components=2,perplexity=5,rando
m_state=42).fit_transform(vectors)
  plt.figure(figsize=(8,6))
  plt.scatter(reduced[:,0],reduced[:,1])
 for i,word in enumerate(words):
    plt.annotate(word,(reduced[i,0],reduced[i,1]),fontsize
=12)
  plt.title(f"Word Embedding Visualization using
{method}")
  plt.show()
plot_embeddings(vectors,words,method="PCA")
plot_embeddings(vectors,words,method="t-SNE")
```

PROGRAM 6

```
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
def analyze_sentiment(text):
  result = sentiment_pipeline(text)[0] # Get the first
result
  label = result["label"]
  confidence = result["score"]
  return f"Sentiment: {label} (Confidence:
{confidence:.2f})"
texts = [
"I love this product! It's amazing.",
"This is the worst experience I've ever had."]
for text in texts:
  print(f"Text: {text}")
  print(analyze_sentiment(text))
  print("-" * 50)
```

PROGRAM 3

```
!pip install --upgrade gensim
!pip install numpy==1.23.5
import gensim
from gensim.models import Word2Vec
import nltk
from nltk.tokenize import word_tokenize
import string
import pandas as pd
nltk.download('all')
df=pd.read_csv("alldata_1_for_kaggle.csv",encoding='I
SO-8859-1')
medical_corpus=df['a'].to_list()
def preprocess_text(corpus):
  processed=[]
  for sentence in corpus:
    tokens=word_tokenize(sentence.lower())
    tokens=[word for word in tokens if word.isalpha()]
    processed.append(tokens)
  return processed
tokenized_corpus=preprocess_text(medical_corpus)
model=Word2Vec(sentences=tokenized_corpus,vector_s
ize=100,window=5,workers=4)
model.save("medical_word2vec.model")
model=Word2Vec.load("medical_word2vec.model")
file_name="data.txt"
with open(file_name,"r") as data:
  medical_corpus=data.readlines()
  medical_corpus
 model.wv['diabetes']
  similarity=model.wv.similarity("diabetes","glucose")
  print("Similarity between 'diabetes' and 'glucose':
",similarity)
  print("Words most similar to 'diabetes':
,model.wv.most_similar("diabetes"))
```

PROGRAM 4

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.corpus import wordnet
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt_tab')  'punkt_tab' data package
def get_similar_words(word, top_n=1):
   """Find similar words using WordNet synonyms."""
   synonyms = set()
   for syn in wordnet.synsets(word):
      for lemma in syn.lemmas():
         synonyms.add(lemma.name().replace("_", " "))
 synonyms.discard(word)
   return list(synonyms)[:top_n]
def enhance_prompt(original_prompt):
   """Enhance the prompt by replacing certain words
with synonyms."""
   words = word_tokenize(original_prompt)  # Tokenize
sentence
   tagged_words = pos_tag(words)  # POS tagging
   enriched_words = []
   for word, tag in tagged_words:
      if tag in ["NN", "NNS", "JJ", "RB"]:
         similar = get_similar_words(word, top_n=1)
         enriched_words.append(similar[0] if similar else
word)
      else:
         enriched_words.append(word)
   return " ".join(enriched_words)
original_prompt = "Describe the impact of artificial
intelligence on healthcare."
enriched_prompt = enhance_prompt(original_prompt)
print("Original Prompt:", original_prompt)
print("Enriched Prompt:", enriched_prompt)
```

PROGRAM 5

```
!pip install sentence-transformers
from sentence_transformers import
SentenceTransformer, util
import torch
import random
model = SentenceTransformer('all-MiniLM-L6-v2')
def get_similar_words(word, word_list, top_n=5):
  word_embeddings = model.encode([word] + word_list,
convert_to_tensor=True)
  similarities =
util.pytorch_cos_sim(word_embeddings[0],
word_embeddings[1:]).squeeze(0)
  top_indices = torch.topk(similarities,
top_n).indices.tolist()
  return [word_list[i] for i in top_indices]
def generate_story(seed_word):
  word_list = ["adventure", "journey", "quest",
"mystery", "discovery", "expedition",
"exploration","voyage"]
similar_words = get_similar_words(seed_word,
word_list, top_n=5)
  random.shuffle(similar_words)
  story_template = (
    f"One day, a {seed_word} set out on a
{similar_words[0]}. "
    f"Along the way, it stumbled upon a
{similar_words[1]} that led to an unexpected
{similar_words[2]}."
    f"Guided by an old {similar_words[3]}, the
{seed_word} finally reached the ultimate
{similar_words[4]}." )
  return story_template
seed_word = "explorer"
story = generate_story(seed_word)
print("\nGenerated Story:\n", story)
```

PROGRAM 8

```
!pip install google-api-python-client==2.100.0
!pip install langchain cohere langchain-community
pydantic google-auth google-auth-oauthlib google-auth-
httplib2
import os
os.environ["COHERE_API_KEY"] =
"YOUR_API_KEY"
from google.colab import drive
drive.mount('/content/drive')

# Check if the drive is mounted
if os.path.exists('/content/drive'):
   print("Google Drive is mounted successfully.")
   file_path = "/content/drive/My
Drive/article_000000.txt"
   # Check if the file exists at the specified path
   if os.path.exists(file_path):
      print(f"File found at: {file_path}")
      with open(file_path, "r") as file:
         document_text = file.read()
      print(document_text[:500])  # Print first 500
characters
   else:
      print(f"Error: File not found at {file_path}")
      print("Please check the file path and ensure the file
exists in your Google Drive 'My Drive' folder.")
else:
   print("Error: Google Drive could not be mounted.")
   print("Please ensure you have authorized Google
Colab to access your Google Drive.")
```

PROGRAM 7

```
from transformers import pipeline
summarizer = pipeline("summarization",
model="facebook/bart-large-cnn")
def summarize_text(text, max_length=130,
min_length=50):
   summary = summarizer(text,
max_length=max_length, min_length=min_length,
do_sample=False)
   return summary[0]["summary_text"]
long_text = """
Artificial Intelligence (AI) is a rapidly advancing field
that aims to create machines capable of human-like
thinking.
AI is used in various industries, from healthcare to
finance, improving efficiency and accuracy. Machine
learning,
a subset of AI, enables computers to learn from data and
make predictions without being explicitly programmed.
With deep learning, neural networks can process vast
amounts of information and recognize patterns, leading
to advancements in self-driving cars, natural language
processing, and medical diagnostics.
"""
summary = summarize_text(long_text)
print("Original Text:")
print(long_text)
print("\nSummarized Text:")
print(summary)
```

PROGRAM 9

```
import os
from langchain.llms import Cohere
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
import wikipediaapi
from pydantic import BaseModel
os.environ["COHERE_API_KEY"] =
"YOUR_API_KEY"
llm = Cohere(model="command")
class Info(BaseModel):
   founder: str; founded_year: str; branches: str;
employees: str; summary: str
def extract(name):
   wiki = wikipediaapi.Wikipedia(user_agent='app',
language='en')
   text = wiki.page(name).summary
      response = LLMChain(
      llm=llm,
      prompt=PromptTemplate(
         input_variables=["text"],
         template="Extract: Founder, Year, Branches,
Employees, Summary from: {text}"
      )
   ).run(text=text)
   lines = [line.split(":", 1)[1].strip() for line in
response.split("\n")]
   return Info(founder=lines[0], founded_year=lines[1],
branches=lines[2], employees=lines[3],
summary=lines[4])
name = input("Institution: ")
print(extract(name).model_dump_json(indent=2))
```

PROGRAM 10

```
!pip install langchain langchain_community cohere
faiss-cpu pypdf
import langchain, cohere
print("LangChain Version:", langchain.__version__)
print("Cohere Version:", cohere.__version__)
!pip install langchain-cohere
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import
RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.chains import RetrievalQA
from langchain.embeddings import
HuggingFaceEmbeddings
from langchain.llms import HuggingFacePipeline
from transformers import AutoTokenizer,
AutoModelForCausalLM, pipeline
import torch
pdf_path = "10th class english 2020-21 20.pdf"  #
Ensure this file is uploaded to Colab
loader = PyPDFLoader(pdf_path)
documents = loader.load()
text_splitter =
RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=100)
texts = text_splitter.split_documents(documents)
embeddings =
HuggingFaceEmbeddings(model_name="sentence-
transformers/all-MiniLM-L6-v2")
vector_store = FAISS.from_documents(texts,
embeddings)
model_id = "tiiuae/falcon-7b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
   model_id,
   torch_dtype=torch.float16,  # Efficient memory use
   device_map="auto"  # Automatically assign
GPU/CPU
)
print("Falcon-7B Loaded Successfully!")
pipe = pipeline(
   "text-generation",
   model=model,
   tokenizer=tokenizer,
   torch_dtype=torch.float16,
   device_map="auto",
   max_new_tokens=512,
   do_sample=True,
   temperature=0.7,
)
llm = HuggingFacePipeline(pipeline=pipe)
qa_chain = RetrievalQA.from_chain_type(llm,
retriever=vector_store.as_retriever())
print("Done all steps successfully")
def chatbot():
   print(" 🔹 IPC Chatbot is ready! (Type 'exit' to stop)")
   while True:
      query = input("You: ")
      if query.lower() == "exit":
         print("Chatbot: Goodbye! 👋")
         break
      response = qa_chain.run(query)
      print(f"Chatbot: {response}")
chatbot()
```