|  |  |
| --- | --- |
| CREATE A CHATBOT USING PYTHON | TEAM MEMBER  NAME :THARUNA SELVI R  REG NO:820421205076  PHASE 3:  Development Part 1 |



SYNOPSIS

* IntRoduction
* SETTING UP THE ENVIRONMENT AND INSTALLING REQUIRED LIBRARIES
* Loading the dataset
* Data preprocessing
* Data Collection
* Data Cleaning
* Data Reduction
* Data Transformation
* Data Discrimination
* GPT3
* FLASK
* CONCLUSION

INTRODUCTION:

Startproject by load and prepare your dataset, configure your development environment, and create basic user interactions. Install necessary libraries like transformers and Flask for easy GPT-3 integration and web application development. This sets the foundation for a complex, user-friendly chatbot interface. As you explore the dataset's complexities, you'll create a dynamic, intelligent chatbot that engages people effectively.

SETTING UP THE ENVIRONMENT AND INSTALLING REQUIRED LIBRARIES:

Starting the set up the environment by installing the necessary

libraries and frameworks. Here I use virtual environments to manage

dependencies. Here's an example of how to set up a virtual environment

and install some essential packages:

# Create a virtual environment

python -m venv chatbot-env

# Activate the virtual environment

source chatbot-env/bin/activate

# On Windows

use "chatbot-env\Scripts\activate"

# Install required libraries

pip install transformers flask nltk

LIBRARIES:

pip install pandas

pip install numpy

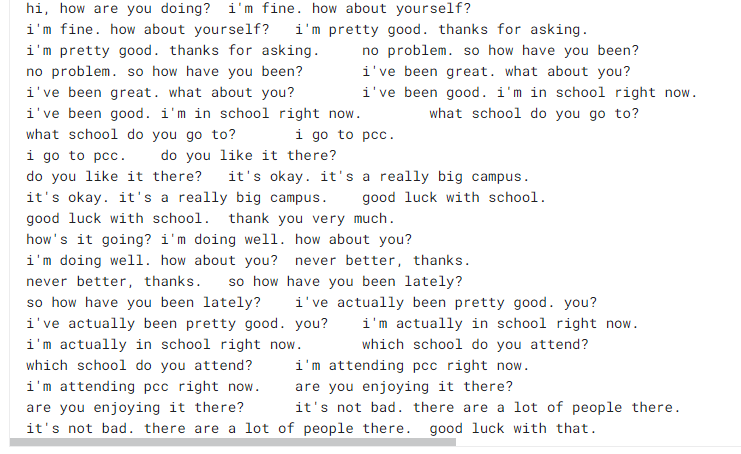
pip install io

pip installnltk

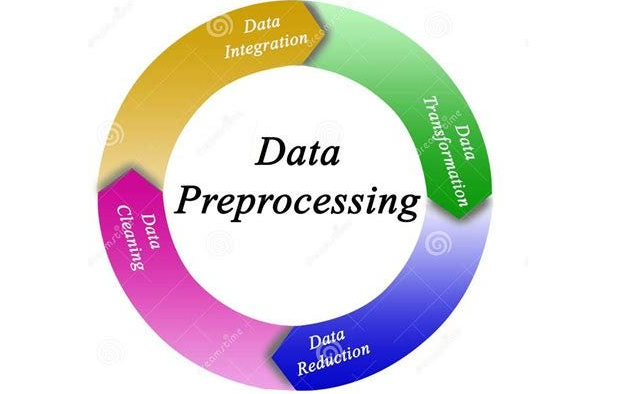
pip install scikit-learn

LOADING THE DATASET:

Dataset Link:[**https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot**](https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot)



DATA PREPROCESSING:

* DEFINITION:
* Data preprocessing is the initial step in data analysis and machine learning.
* Data Preprocessing is a process to convert the raw data into meaningful data using different techniques.
* **IMPORTANCE:**
* Data Quality Improvement
* Handling Missing Data
* Scaling and Normalization
* Categorical Data Handling
* Time and Resource Efficiency
* TECHNIQUES:
* Data Collection
* Data Cleaning
* Data Reduction
* Data Transformation
* Data Discrimination
* DATA COLLECTION:

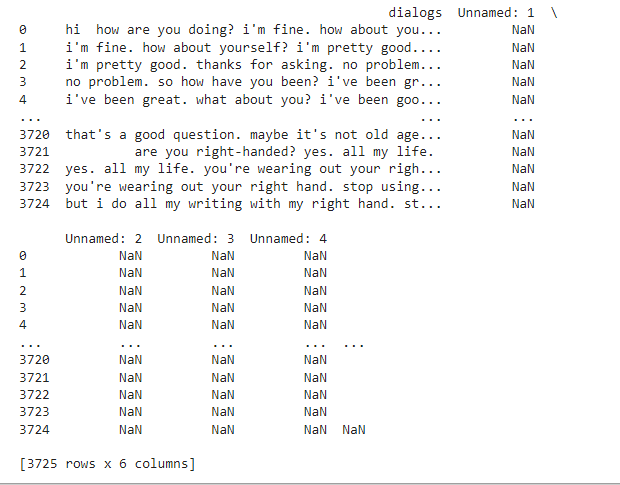
Data collection involves gathering and analyzing information from various sources to solve research issues, answer questions, evaluate results, and anticipate trends in various fields like business, and healthcare.

import pandas as pd

import io

df = pd.read\_csv(io.BytesIO(uploaded['dialog1.csv']))

print(df)





df.head()

df.shape()

33.PNG

* DATA CLEANING:
* DEFINITION:

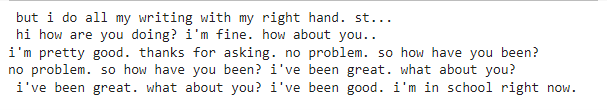
Data cleaning means fill in missing values ,smooth out noise while identifying outliers and correct inconsistencies data

**LOWERCASING:**

m\_str=" But I do all my writing with my right hand. st...\n Hi how are you doing? i'm fine. How about you..\ni'm pretty good. Thanks for as69king. no Problem. so How have you been? \nno problem. So how have you been? I've been great. what about you? \n I've been great. what about you? I've been good. I'm in school right now. "

text=m\_str.lower()

print(text)

****

**REMOVING PUNCTUATION:**

punc='''!()-[]{};:'"\,<>./?@#$%^&\*\_~'''

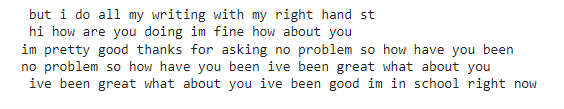
no\_punct=""

for char in text:

if(char not in punc):

no\_punct=no\_punct+char

print(no\_punct)

****

**REMOVING STOPWARD:**

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

stop=stopwords.words("english")

text\_cleaned=""

for word in text.split():

if word in stop:

pass

else:

text\_cleaned +=" "

text\_cleaned +=word

text\_cleaned

**7.PNG**

**LEMMATIZATION:**

import nltk

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lem=WordNetLemmatizer()

text\_cleaned1=" "

for word in text\_cleaned.split():

word=lem.lemmatize(word,pos="v")

text\_cleaned1 +=" "

text\_cleaned1 +=word

print(text\_cleaned1)

**7.PNG**

**STEMMING:**

text=text\_cleaned.strip()

print(text)

**8.PNGTOKENIZATION:**

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

text\_dataset = [" but i do all my writing with my right hand stand",

" hi how are you doing im fine how about you",

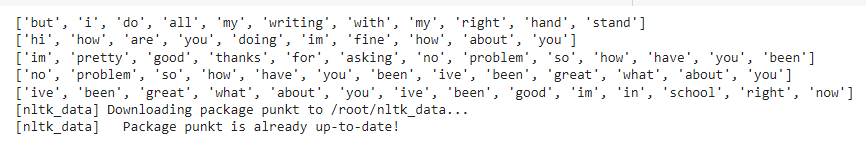
"im pretty good thanks for asking no problem so how have you been ",

"ive been great what about you ive been good im in school right now "]

tokenized\_dataset = [word\_tokenize(sentence)for sentenceintext\_dataset]

for tokens in tokenized\_dataset:

print(tokens)

****

**HANDLING DUPLICATES:**

import pandas as pd

import io

def remove\_duplicates\_with\_set(text\_list):

unique\_text\_set = set()

result = []

for text in text\_list:

if text not in unique\_text\_set:

unique\_text\_set.add(text)

result.append(text)

return result

def remove\_duplicates\_by\_comparison(text\_list):

for text in text\_list:

if text not in result:

result.append(text)

return result

text\_dataset =pd.read\_csv(io.BytesIO(uploaded['dialog1.csv']))

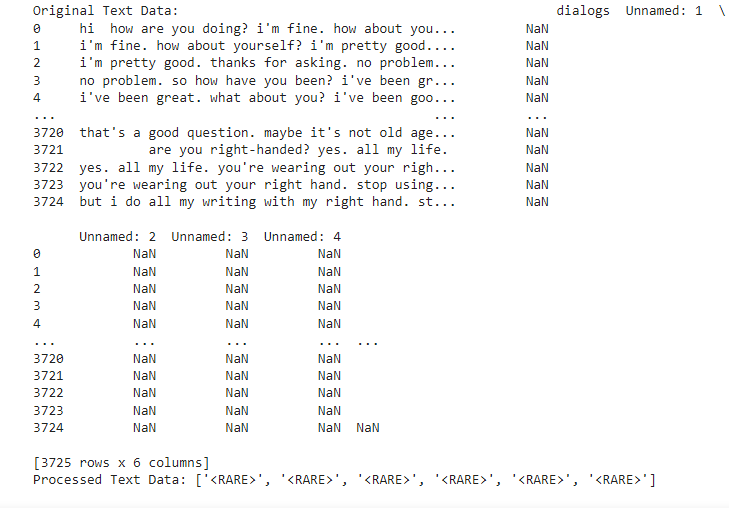
unique\_texts\_set = remove\_duplicates\_with\_set(text\_dataset)

print("Method 1 - Using a Set:", unique\_texts\_set)

unique\_texts\_comparison = remove\_duplicates\_by\_comparison(text\_dataset)

print("Method 2 - By Comparison:", unique\_texts\_comparison)

result = []

****

**IGNORE MISSING DATA:**

import pandas as pd

import numpy as np

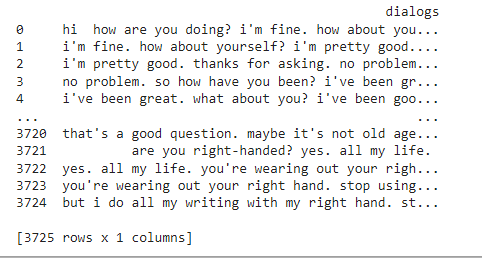
import io

df= pd.read\_csv(io.BytesIO(uploaded['dialog1.csv']))

df.isnull()

text\_data=df.dropna(axis = 1)

print(text\_data)

****

* **DATA REDUCTION:**

**DEFINTION:**

Data reduction isaiming to reduce complexity while retaining essential information. This reduces computational resources, improves training efficiency, and minimizes noise

**DIMENSIONAL REDUCTION:**

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

documents = [" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you", "im pretty good thanks for asking no problem so how have you been ", "no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(documents)

n\_components = 2

svd = TruncatedSVD(n\_components=n\_components)

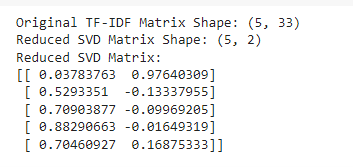
svd\_matrix = svd.fit\_transform(tfidf\_matrix)

print("Original TF-IDF Matrix Shape:", tfidf\_matrix.shape)

print("Reduced SVD Matrix Shape:", svd\_matrix.shape)

print("Reduced SVD Matrix:")

print(svd\_matrix)

**HANDLE RARE WORD:**

import pandas as pd

import io

from collections import Counter

def handle\_rare\_words(text\_data, threshold=2, rare\_token="<RARE>"):

word\_counts = Counter(text\_data)

rare\_words = [word for word, count in word\_counts.items() if count <= threshold]

processed\_text = [rare\_token if word in rare\_words else word for word in text\_data]

return processed\_text

threshold = 2

processed\_text = handle\_rare\_words(text\_data, threshold)

print("Processed Text Data:", processed\_text)

****

**REGRESSION:**

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

data = [" but i do all my writing with my right hand stand", " hi how are you doing im fine how about you", "im pretty good thanks for asking no problem so how have you been ", "no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

target = [3, 4, 1, 5,2]

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(data)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, target, test\_size=0.2, random\_state=42)

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

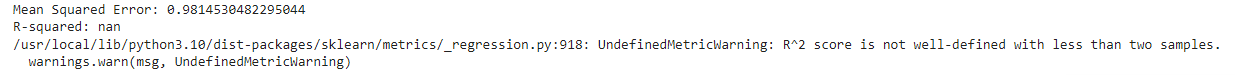
predictions = regressor.predict(X\_test)

mse = mean\_squared\_error(y\_test, predictions)

r2 = r2\_score(y\_test, predictions)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

**CLUSTER:**

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.metrics import adjusted\_rand\_score

import pandas as pd

data = [" but i do all my writing with my right hand stand", " hi how are you doing im fine how about you","im pretty good thanks for asking no problem so how have you been ","no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(data)

num\_clusters = 2

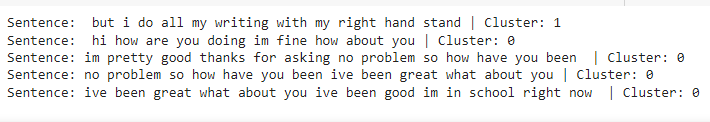
kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

kmeans.fit(X)

cluster\_labels = kmeans.labels\_

for i, sentence in enumerate(data):

print(f"Sentence: {sentence} | Cluster: {cluster\_labels[i]}")

****

**VECTORIZATION:**

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from gensim.models import Word2Vec

from nltk.tokenize import word\_tokenize

import nltk

nltk.download('punkt')

data = [" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you","im pretty good thanks for asking no problem so how have you been ","no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(data)

vectorized\_data = X.toarray()

print("Vectorized Data (using CountVectorizer):\n", vectorized\_data)

tokenized\_data = [word\_tokenize(sentence.lower()) for sentence in data]

word2vec\_model = Word2Vec(sentences=tokenized\_data, vector\_size=100, window=5, min\_count=1, sg=0)

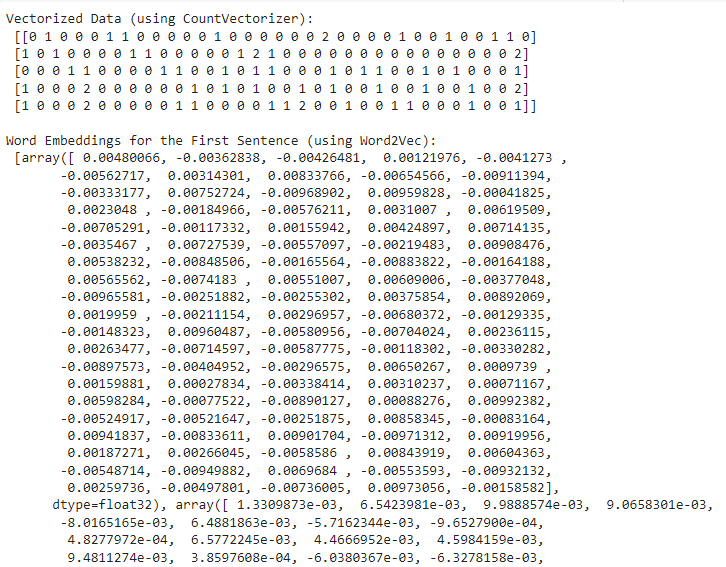
first\_sentence\_embeddings = []

for word in tokenized\_data[0]:

if word in word2vec\_model.wv:

first\_sentence\_embeddings.append(word2vec\_model.wv[word])

print("\nWord Embeddings for the First Sentence (using Word2Vec):\n", first\_sentence\_embeddings)

****

* **DATA TRANSFORMATION:**

**DEFINITION:**

Data transformation is a crucial step in preprocessing a chatbot dataset, converting, modifying, or structuring data for analysis, model training, and interaction with the chatbot.

ATTRIBUTE SELECTION:

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

data = [

" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you","im pretty good thanks f or asking no problem so how have you been ","no problem so how have you been ive been great what about you" ,"ive been great what about you ive been good im in school right now "

]

df = pd.DataFrame(data, columns=["text"])

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(df['text'])

feature\_names = tfidf\_vectorizer.get\_feature\_names\_out()

tfidf\_df = pd.DataFrame(data=tfidf\_matrix.toarray(), columns=feature\_names)

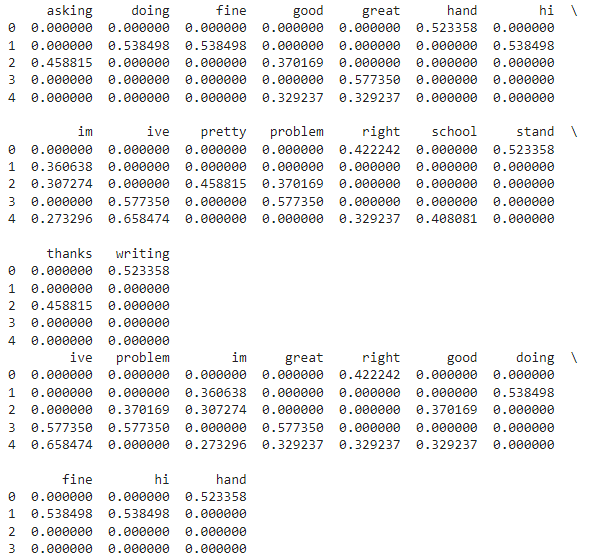
print(tfidf\_df)

top\_n\_features = 10

selected\_features = tfidf\_df.sum().nlargest(top\_n\_features).index

selected\_features\_df = tfidf\_df[selected\_features]

print(selected\_features\_df)

CONCEPT HIERARCHY GENERATION:

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.decomposition import LatentDirichletAllocation

text\_data= [

" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you","im pretty good thanks f or asking no problem so how have you been ","no problem so how have you been ive been great what about you" ,"ive been great what about you ive been good im in school right now "

]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(text\_data)

num\_topics = 2

lda = LatentDirichletAllocation(n\_components=num\_topics, random\_state=42)

lda.fit(X)

feature\_names = vectorizer.get\_feature\_names\_out()

for topic\_idx, topic in enumerate(lda.components\_):

print(f"Topic {topic\_idx + 1}:")

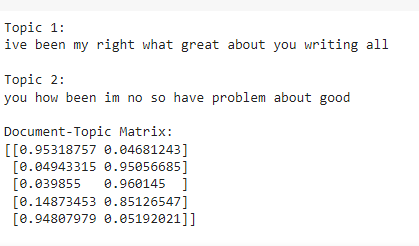
print(" ".join([feature\_names[i] for i in topic.argsort()[:-10 - 1:-1]]))

print()

doc\_topic\_matrix = lda.transform(X)

print("Document-Topic Matrix:")

print(doc\_topic\_matrix)



FEATURE ENGINEERING:

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from gensim.models import Word2Vec

from nltk.tokenize import word\_tokenize

import nltk

nltk.download('punkt')

text\_data= [

" but i do all my writing with my right hand stand", " hi how are you doing im fine how about you","im pretty good thanks f or asking no problem so how have you been ", "no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

labels = [1, 2, 3, 1,2]

tokenized\_text = [word\_tokenize(text) for text in text\_data]

count\_vectorizer = CountVectorizer()

count\_features = count\_vectorizer.fit\_transform([' '.join(tokens) for tokens in tokenized\_text])

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_features = tfidf\_vectorizer.fit\_transform([' '.join(tokens) for tokens in tokenized\_text])

word2vec\_model = Word2Vec(sentences=tokenized\_text, vector\_size=100, window=5, min\_count=1, sg=0)

word\_embeddings = np.array([np.mean([word2vec\_model.wv[word] for word in tokens], axis=0) for tokens in tokenized\_text])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(word\_embeddings, labels, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print("Count Vectorization Features:")

print(count\_features.toarray())

print("TF-IDF Features:")

print(tfidf\_features.toarray())

print("Word Embeddings:")

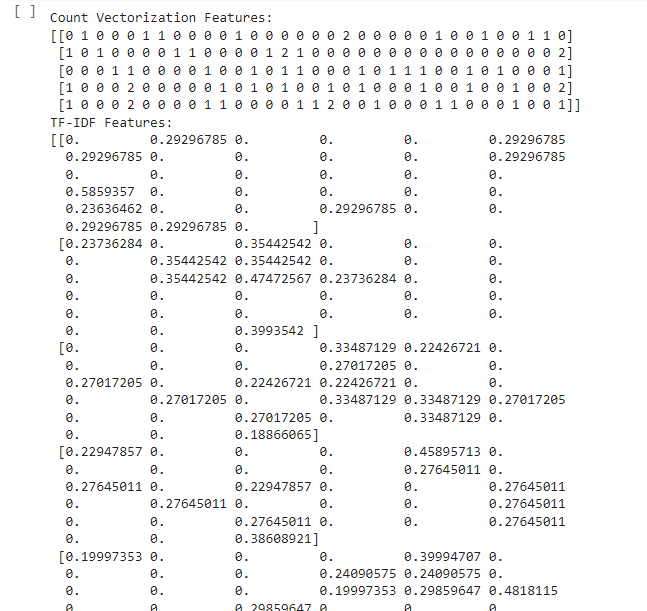
print(word\_embeddings)

print("X\_train\_scaled:")

print(X\_train\_scaled)

print("X\_test\_scaled:")

print(X\_test\_scaled)



* DATA DISCRETIZATION:

Data discretization is the process of dividing continuous or numerical data into distinct intervals, bins, or categories. This simplifies analysis, interpretation, and use in chatbot interactions.

TEXT CATEGORIZATION:

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

text\_data = [

" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you", "im pretty good thanks f or asking no problem so how have you been ", "no problem so how have you been ive been great what about you" , "]

labels = ['positive', 'negative', 'positive', 'negative', 'positive']

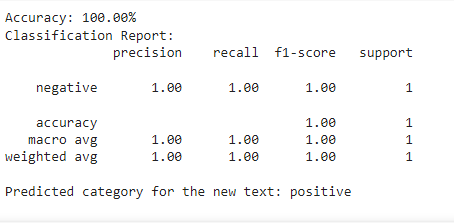
vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(text\_data)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2, random\_state=42)

classifier = MultinomialNB()

classifier.fit(X\_train, y\_train)



DECISION TREE:

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

texts =[

" buti do all my writing with my right hand stand"," hi how are you doing im fine how about you","i was sick. how were you sick? ","no problem so how have you been ive been great what about you" , "my bad i had chores to do. that's all right."]

labels = ["Positive", "Negative", "Neutral", "Positive", "Negative"]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(texts)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

predictions = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, predictions)

report = classification\_report(y\_test, predictions)

print("Accuracy:", accuracy)

print("Classification Report:")

print(report)



TOPIC MODELING:

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from gensim import corpora, models

import gensim

import string

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

text\_data = [

" but i do all my writing with my right hand stand"," hi how are you doing im fine how about you","im pretty good thanks f or asking no problem so how have you been ", "no problem so how have you been ive been great what about you" , "ive been great what about you ive been good im in school right now "]

def preprocess\_text(text):

tokens = word\_tokenize(text.lower())

tokens = [word for word in tokens if word.isalpha() and word not in stopwords.words('english')]

lemmatizer = WordNetLemmatizer()

tokens = [lemmatizer.lemmatize(word) for word in tokens]

return tokens

processed\_text\_data = [preprocess\_text(doc) for doc in text\_data]

dictionary = corpora.Dictionary(processed\_text\_data)

corpus = [dictionary.doc2bow(doc) for doc in processed\_text\_data]

num\_topics = 2

lda\_model = gensim.models.LdaModel(corpus, num\_topics=num\_topics, id2word=dictionary, passes=15)

topics = lda\_model.print\_topics(num\_words=5)

for topic in topics:

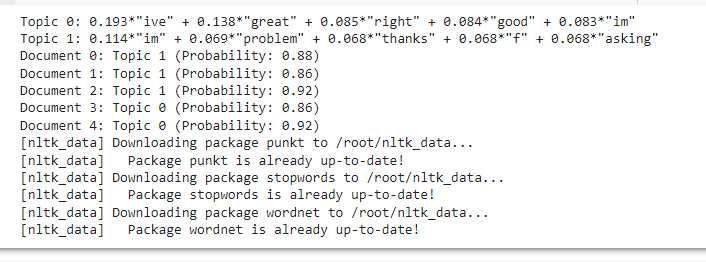
print("Topic {}: {}".format(topic[0], topic[1]))

for i, doc in enumerate(corpus):

topic = lda\_model.get\_document\_topics(doc)

dominant\_topic = sorted(topic, key=lambda x: x[1], reverse=True)[0]

print("Document {}: Topic {} (Probability: {:.2f})".format(i, dominant\_topic[0], dominant\_topic[1]))



DATA VISUALIZATION:

* DATA CLEANING:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

plt.title('Before Cleaning')

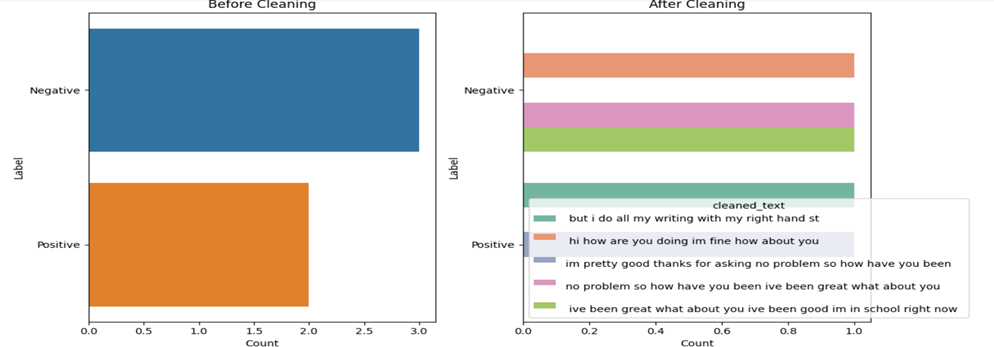
plt.xlabel('Count')

plt.title('After Cleaning')

plt.xlabel('Count')

plt.tight\_layout()

plt.show()



df['questiontokens']=df['question'].apply(lambda x:len(x.split()))

df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

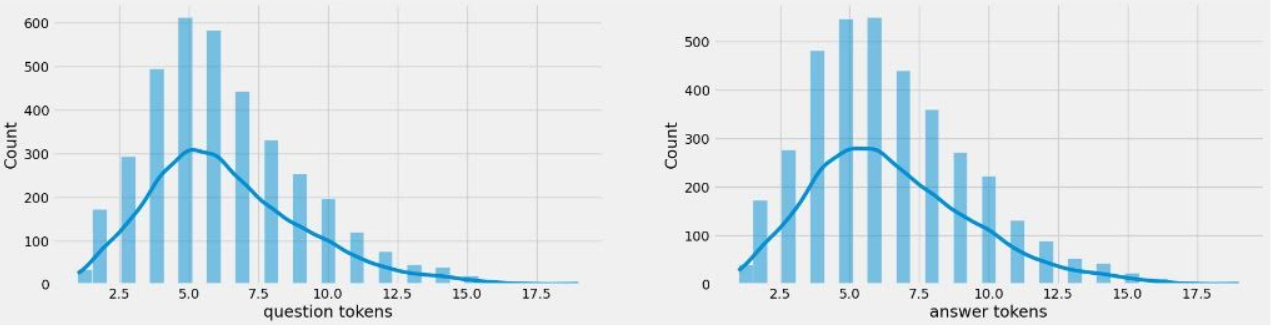
plt.style.use('fivethirtyeight')

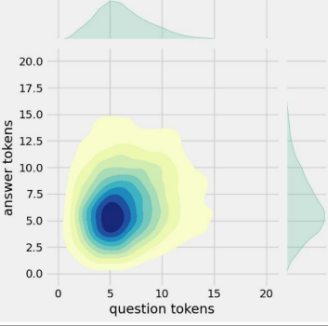
fig,ax=plt.subplots(nrows=1,ncols=2,

figsize=(205))

sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

plt.show()





* DATA REDUCTION:

import pandas as pd

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

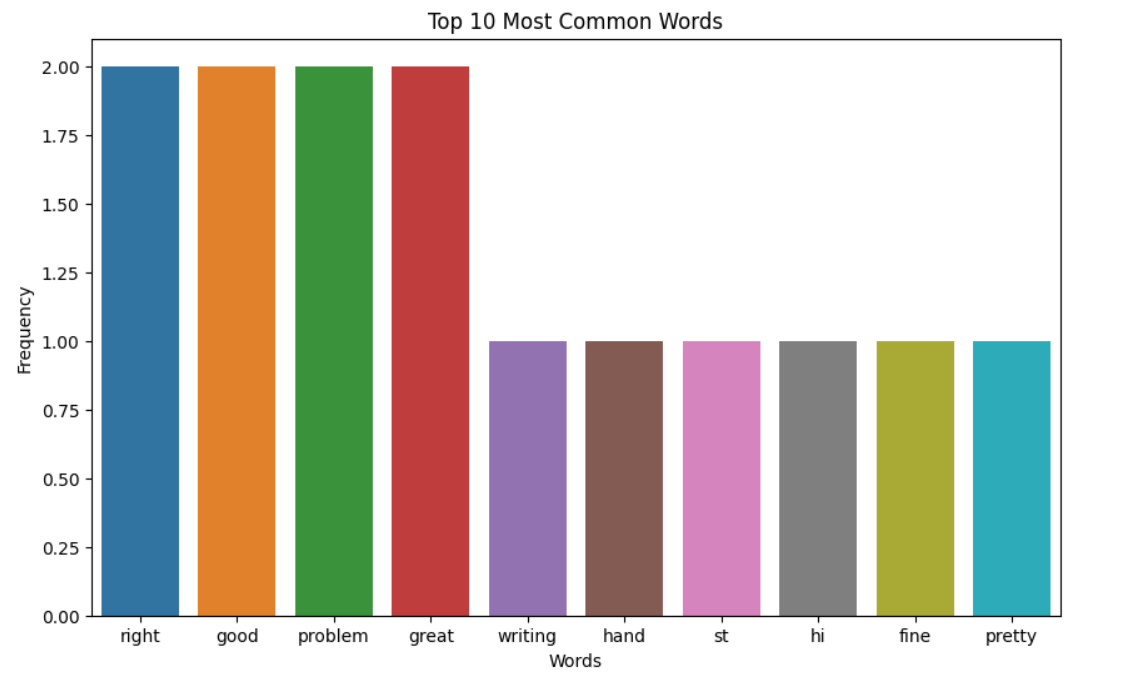
sns.barplot(x=[word[0] for word in common\_words], y=[word[1] for word in common\_words])

plt.xlabel('Words')

plt.ylabel('Frequency')

plt.title('Top 10 Most Common Words')

plt.show()



* DATA TRANSFORMATION:

import matplotlib.pyplot as plt

import numpy as np

words, counts = zip(\*word\_counts.most\_common(20))

plt.bar(words, counts)

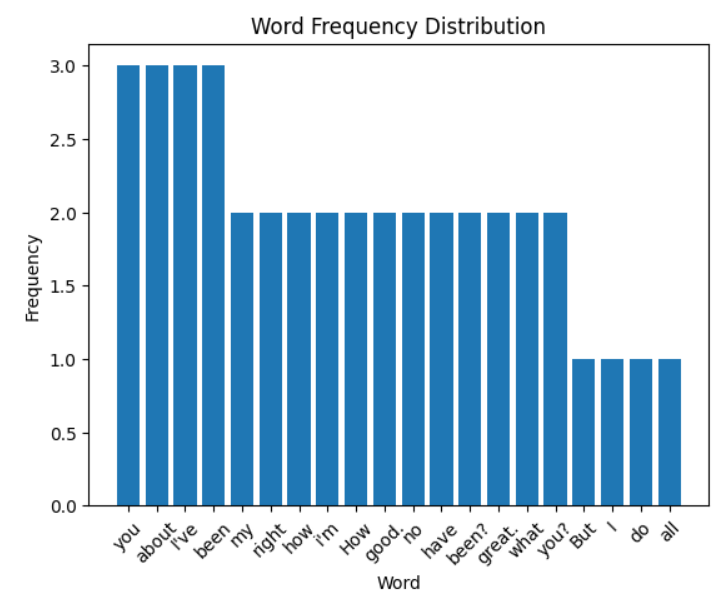
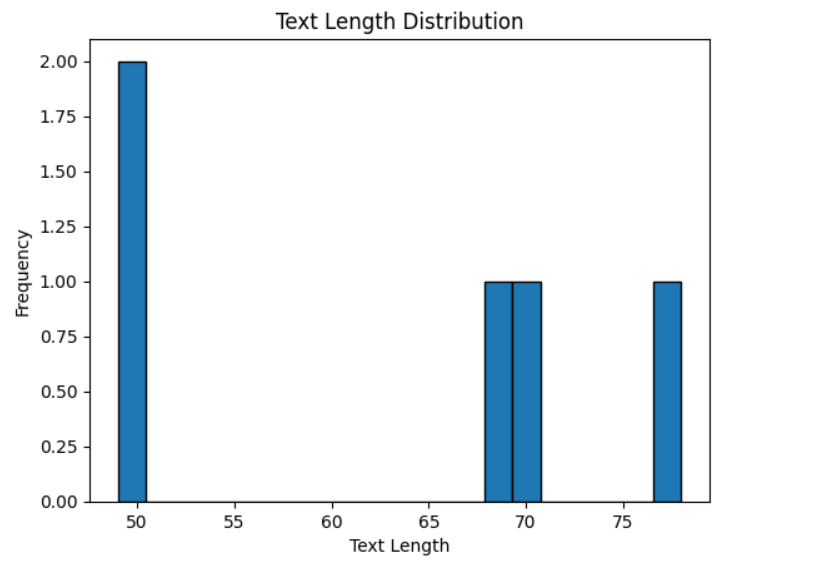
plt.title("Word Frequency Distribution")

plt.xlabel("Word")

plt.ylabel("Frequency")

plt.xticks(rotation=45)

plt.show()



* DATA DISCRETIZATION:

import matplotlib.pyplot as plt

import pandas as pd

from wordcloud import WordCloud

df["Document Length"] = df["Text"].apply(len)

plt.figure(figsize=(8, 5))

plt.bar(df.index, df["Document Length"])

plt.xlabel("Document Index")

plt.ylabel("Document Length")

plt.title("Document Length Distribution")

word\_counts = df["Text"].str.split().apply(len)

plt.figure(figsize=(8, 5))

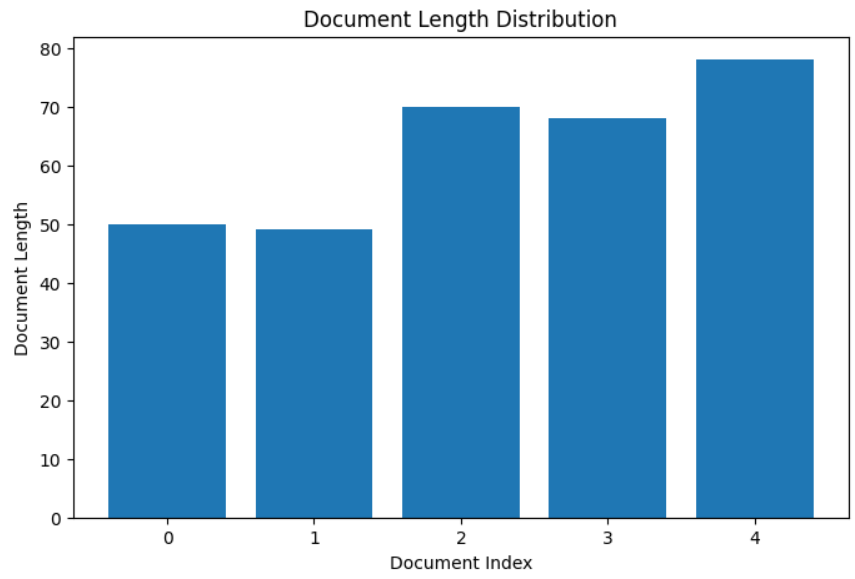
plt.hist(word\_counts, bins=range(0, max(word\_counts) + 1), rwidth=0.8, alpha=0.7)

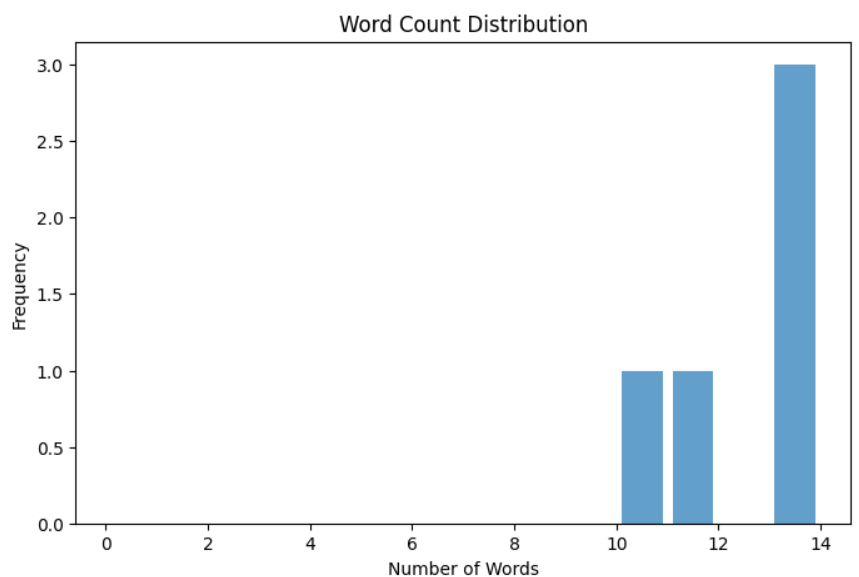
plt.xlabel("Number of Words")

plt.ylabel("Frequency")

plt.title("Word Count Distribution")

plt.show()





GPT3:

Libraries like transformers for GPT-3 integration was not freely accessible to the public, and access to the model was provided by OpenAI through an API on a paid basis.So here I am using the GPT-2.

GPT-2(Generative Pre-trained Transformer 2)isa advanced natural language processing model that can understand and generate human-like text for various tasks. It can be fine-tuned for specific tasks, making it suitable for chatbots, content generation, summarization, and translation. However, it is not perfect and may generate incorrect or biased information.

from transformers import GPT2LMHeadModel, GPT2Tokenizer

model\_name = "gpt2" # You can use other models as well

tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)

model = GPT2LMHeadModel.from\_pretrained(model\_name)

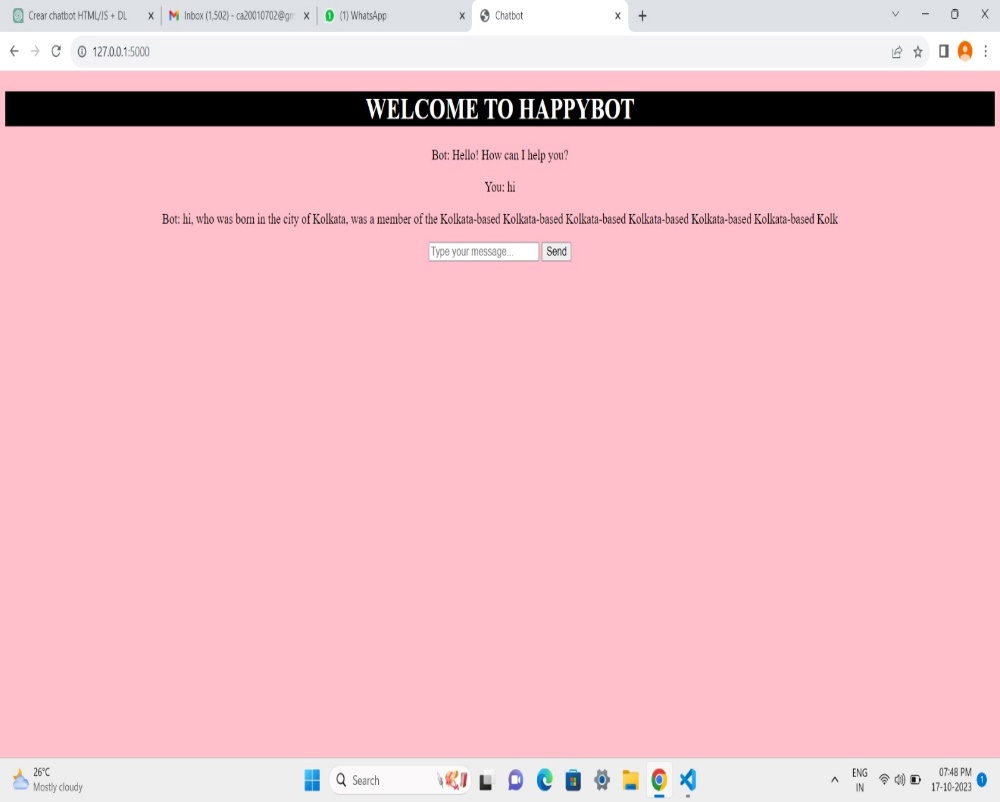
def generate\_response(user\_message):

input\_ids = tokenizer.encode(user\_message, return\_tensors='pt')

response\_ids = model.generate(input\_ids, max\_length=50, num\_return\_sequences=1)

response\_text = tokenizer.decode(response\_ids[0], skip\_special\_tokens=True)

return response\_text



FLASK:

Flask is a Python web framework designed for developers to build web applications, APIs, and interactive services with minimal effort. It provides basic tools for routing, handling requests, and rendering web pages. Flask is often used in combination with other Python libraries to create chatbot interfaces and web applications, allowing users to interact with chatbots through a web browser.

from flask import Flask, request, jsonify

app = Flask(\_name\_)

@app.route('/')

def index():

return open('index.html', 'r').read()

@app.route('/ask', methods=['POST'])

def ask():

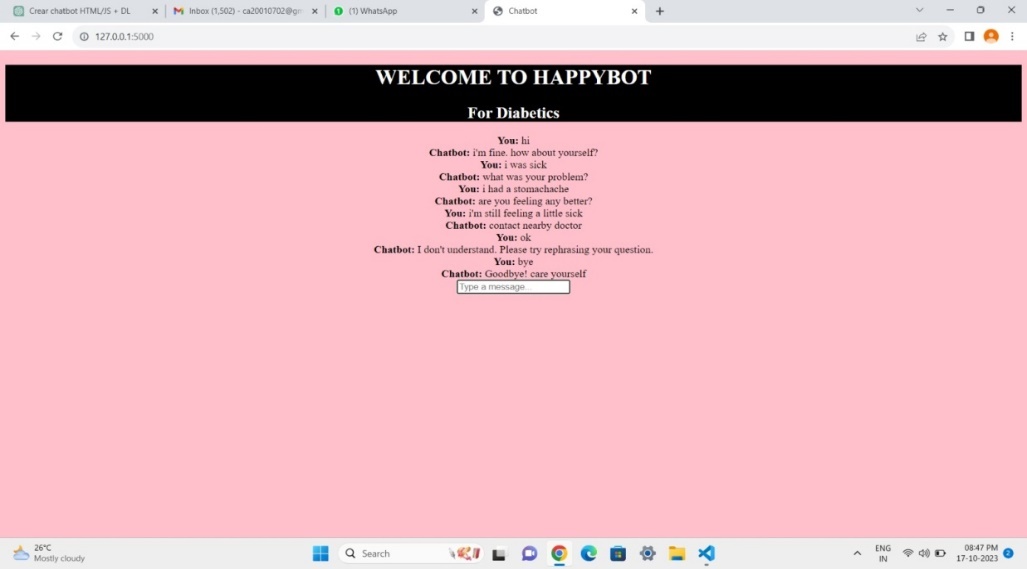
user\_message = request.json.get('userMessage', '')

response = chatbot\_response(user\_message)

return jsonify({'response': response})

if \_name\_ == '\_main\_':

app.run(debug=True)



**CONCLUSION:**

Loading and preprocessing the dataset is crucial for creating a sophisticated chatbot. This involves meticulous preparation of the environment and incorporating user interactions. Installing necessary libraries like transformers for GPT-3 integration and Flask for web app development streamlines the process and equips you with tools for a seamless user experience, ensuring a successful and engaging project.