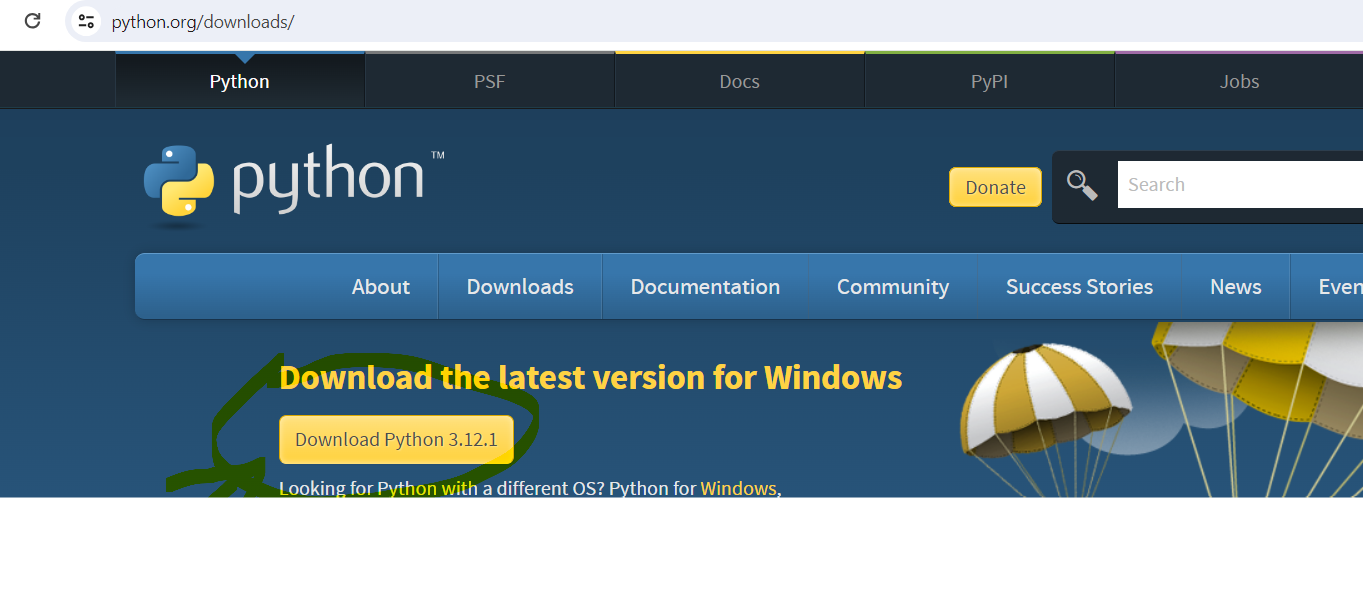
1. **AIM : INSTALLATION OF PYTHON 3 AND PYCHARM**

**Detailed steps to install Python 3 and PyCharm IDE along with NLP library like spaCy**

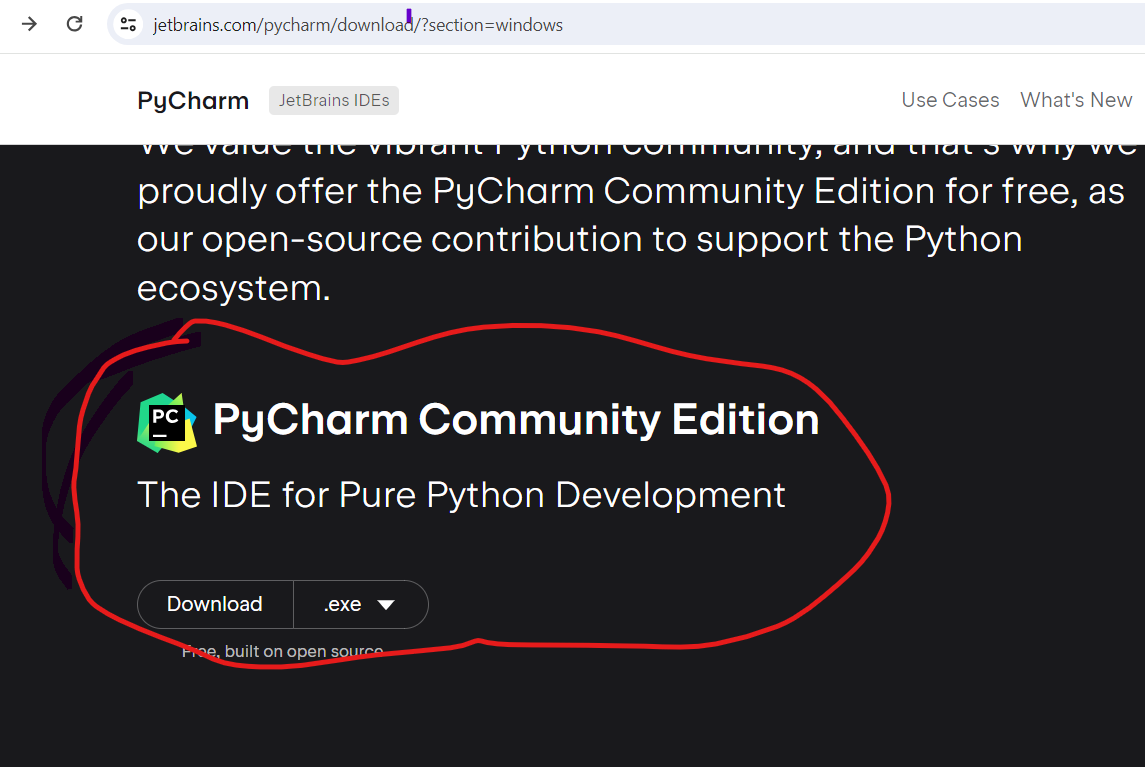
Download & Install Python 3 from below link and click on **Download Python 3**

<https://www.python.org/downloads/>

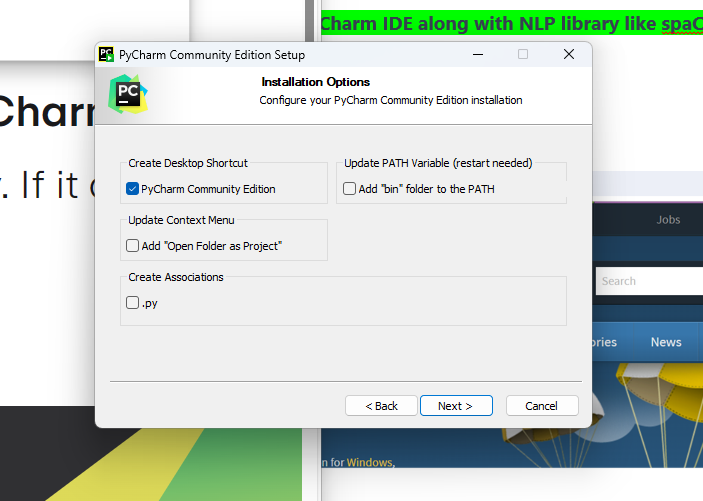


Download and install PyCharm community version from below link

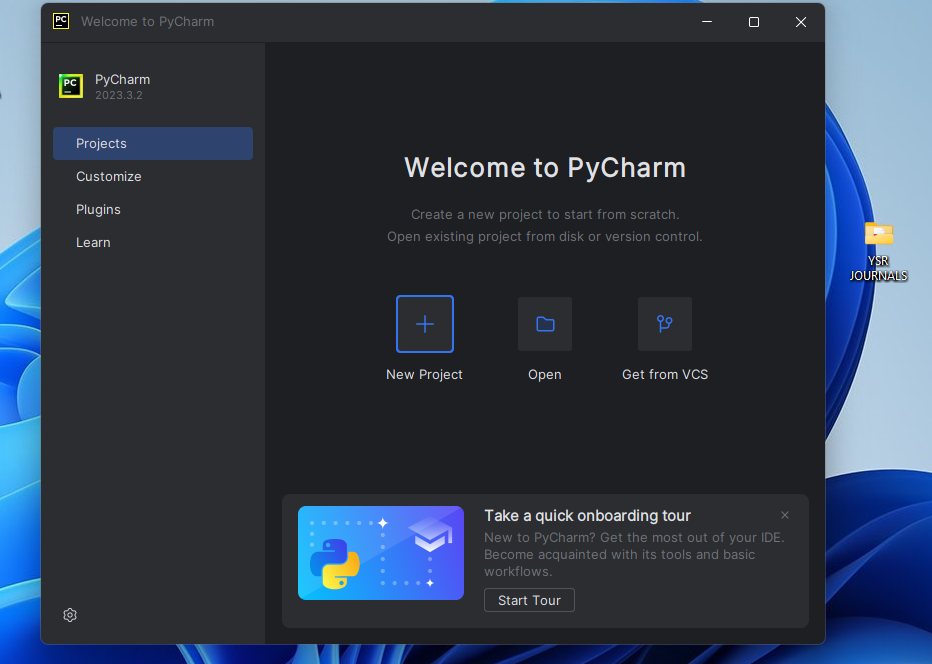
<https://www.jetbrains.com/pycharm/download/?section=windows>



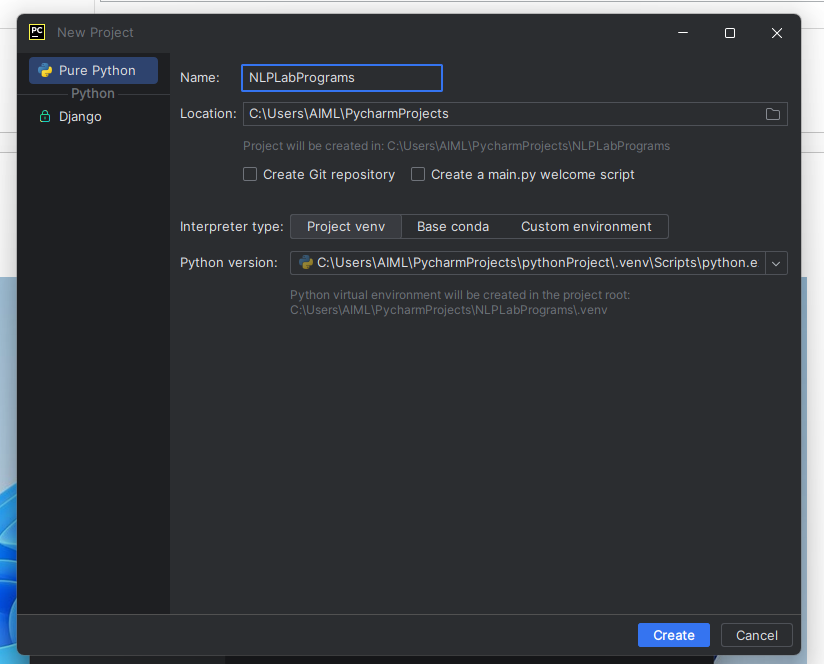
While installing add short cut below

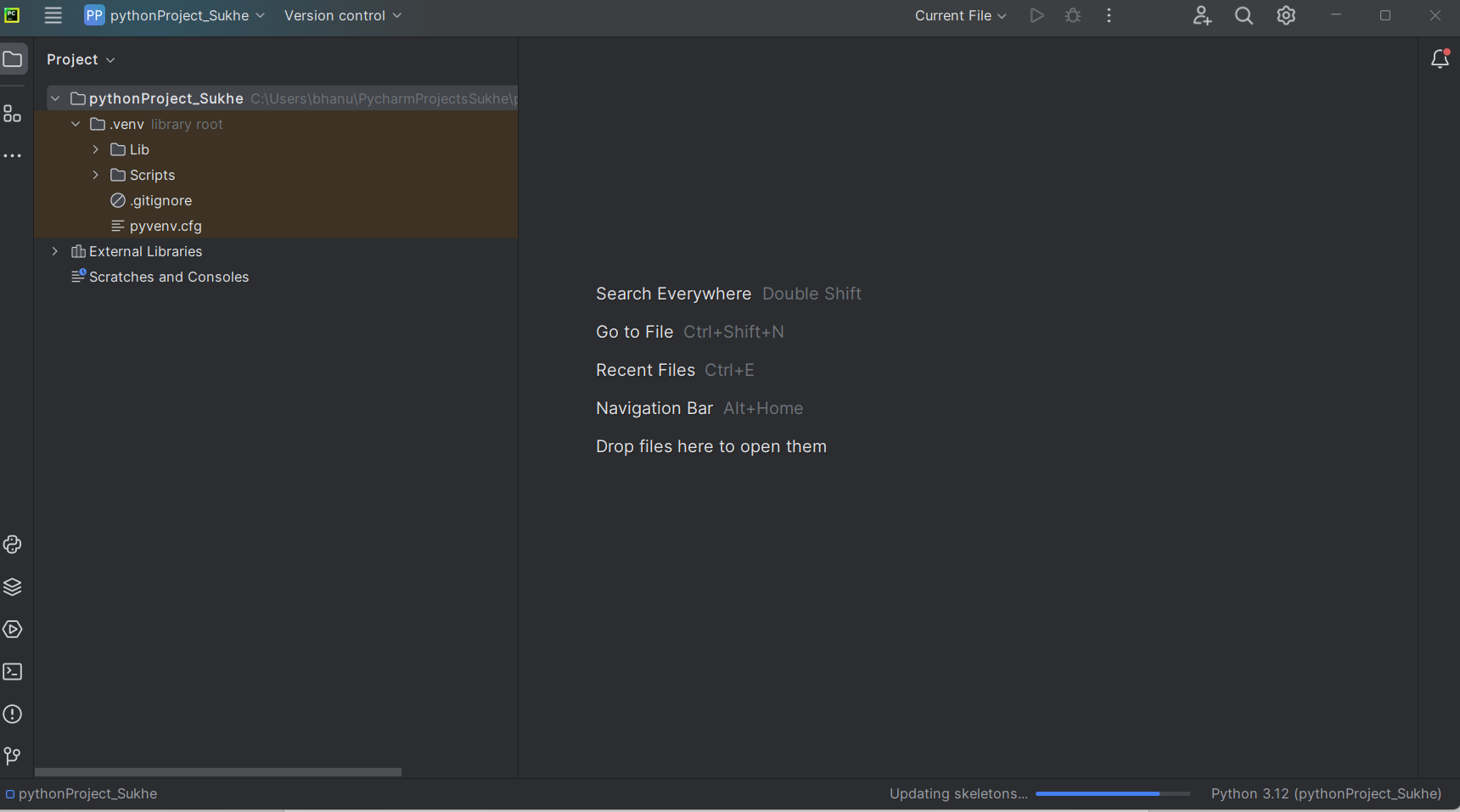


Open PyCharm IDE and create new project and it should look like this

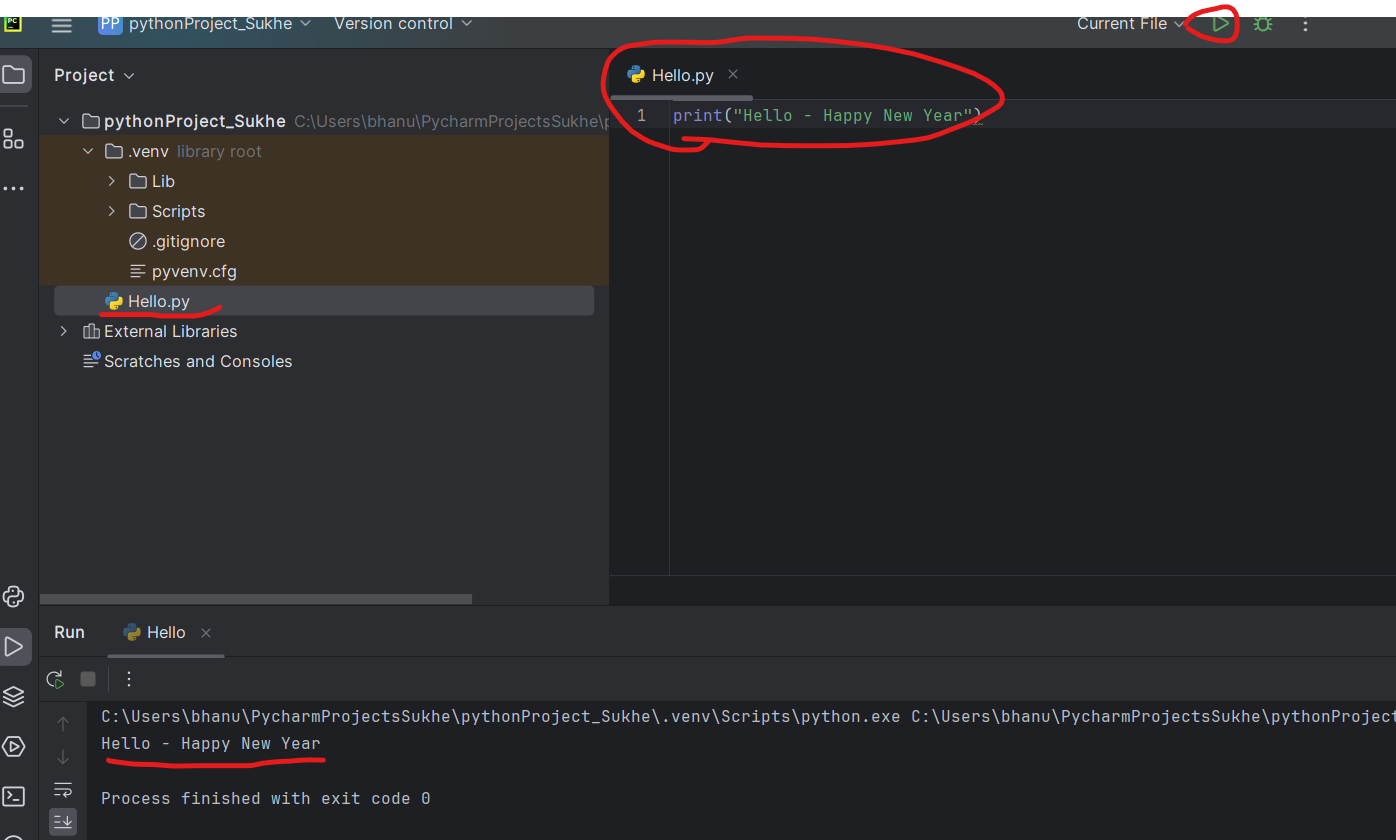


Select New Project and Type NLPLabPrograms



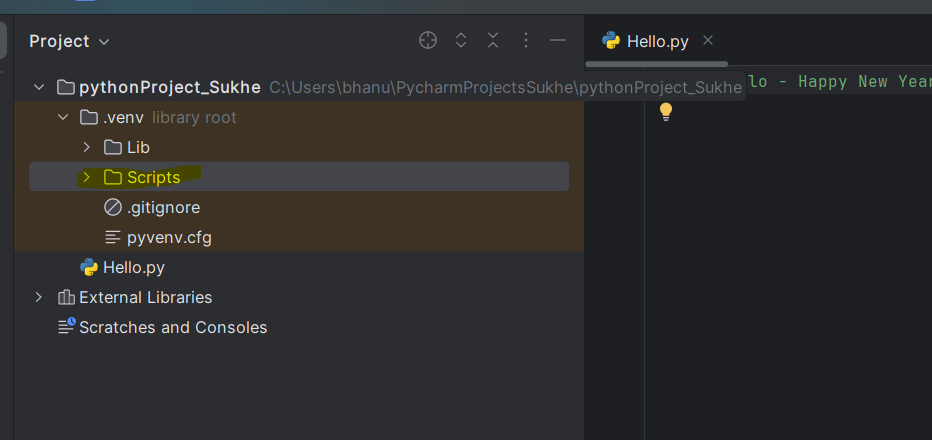


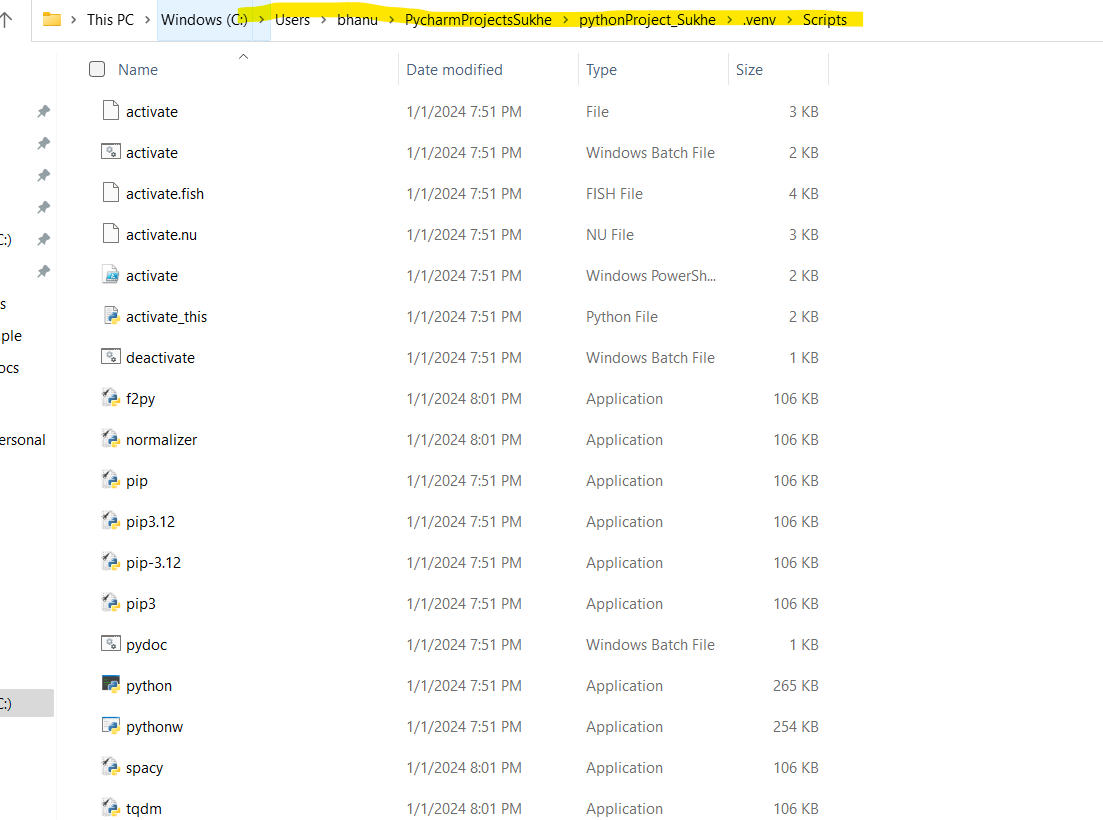
Right click on pythonProject and create new Python File (Hello for example) and write simple print metho and click on Run ( Play Green button on the top right corner) to ensure Python is successfully installed and able to run programs successfully.

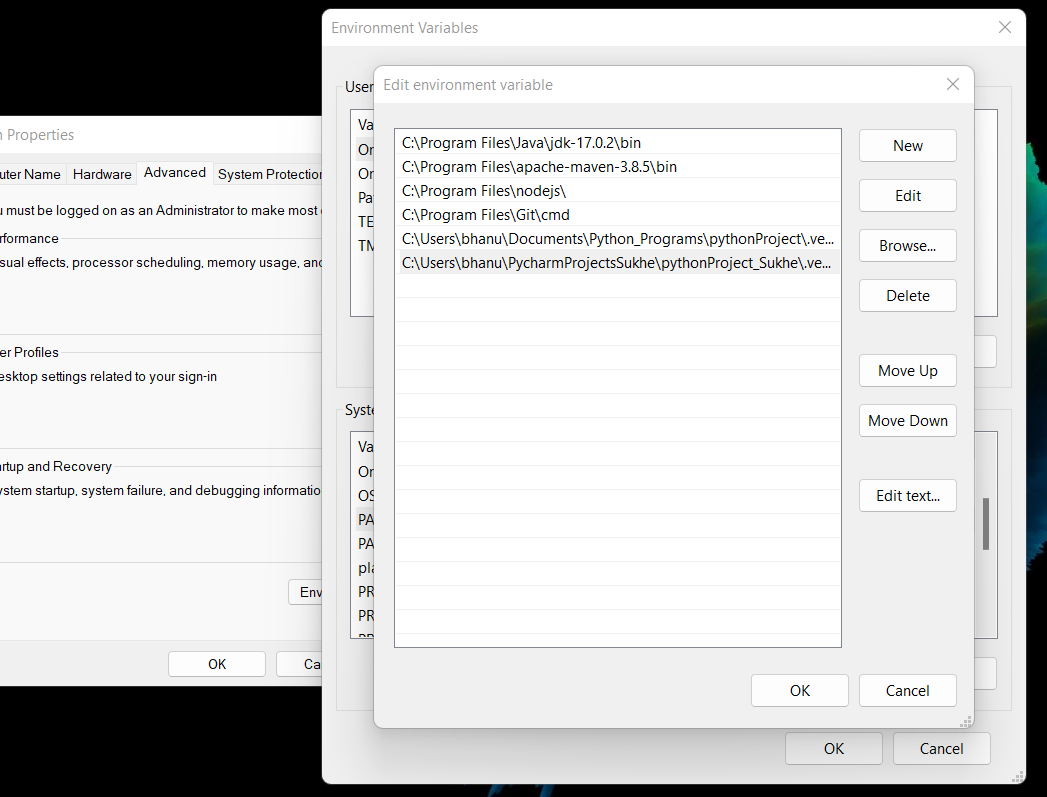


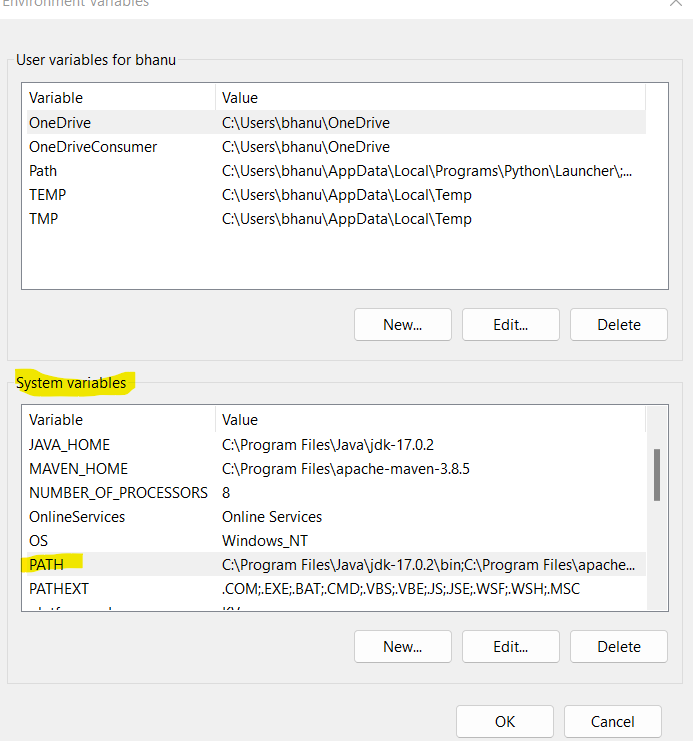
Add Scripts folder path to ENV VARIABLE

Goto System Variables and Edit PATH and add path of Scripts folder









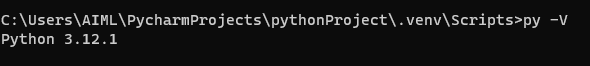
Goto Command prompt and pip version with below command

pip --version



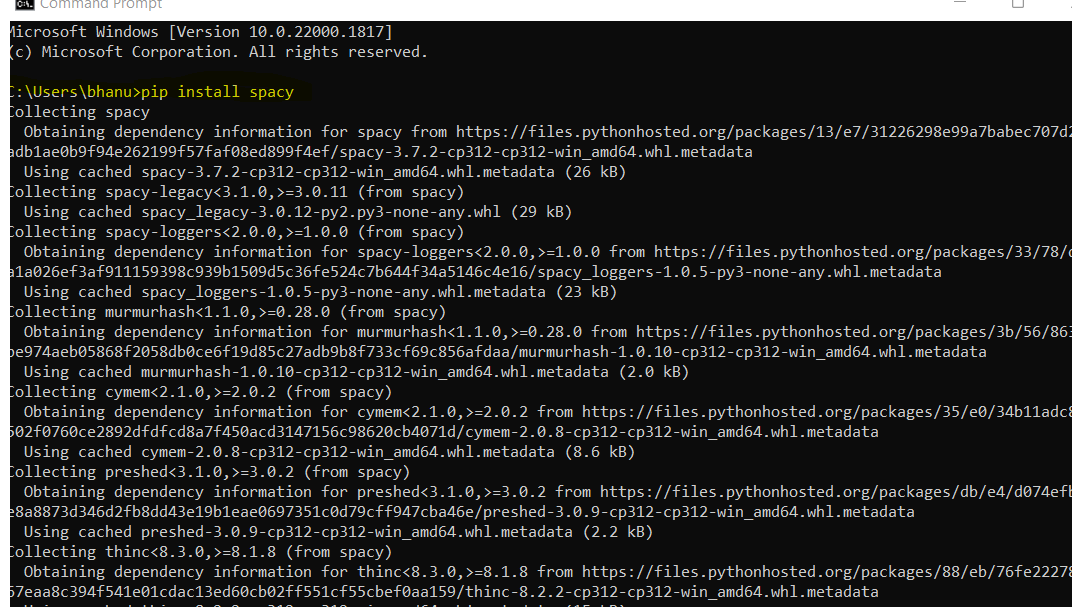
Or

Run Command : py -V

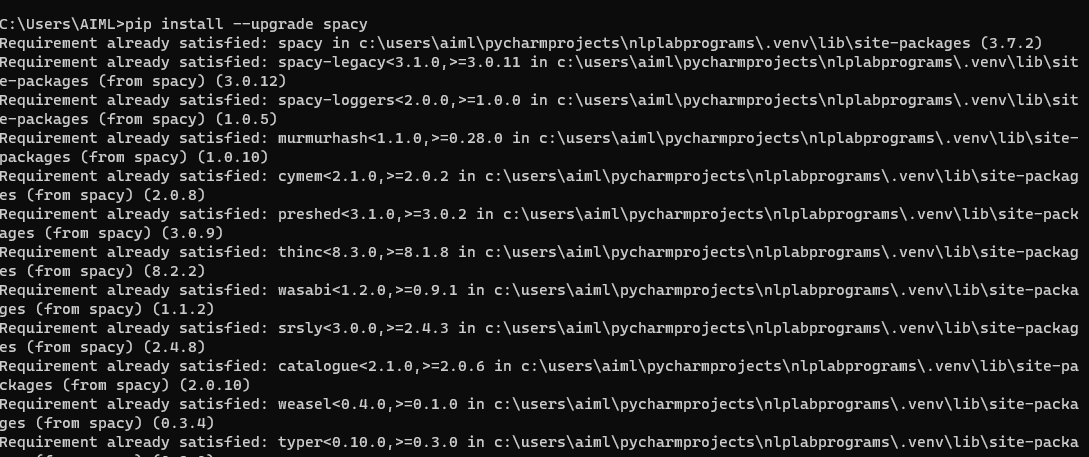


Then provide below command to install SpaCy library for NLP

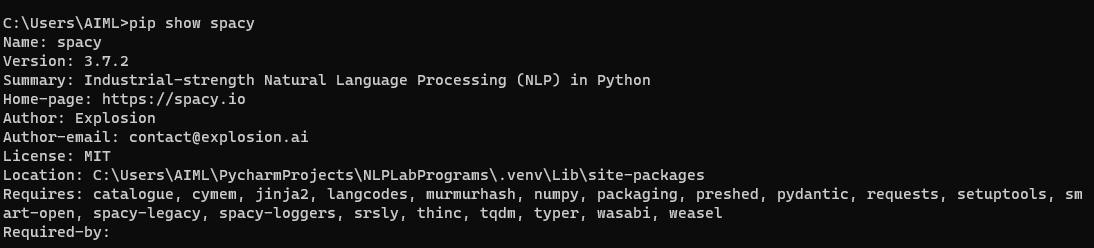
pip install spacy



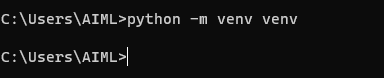
Run Command : pip install --upgrade spacy



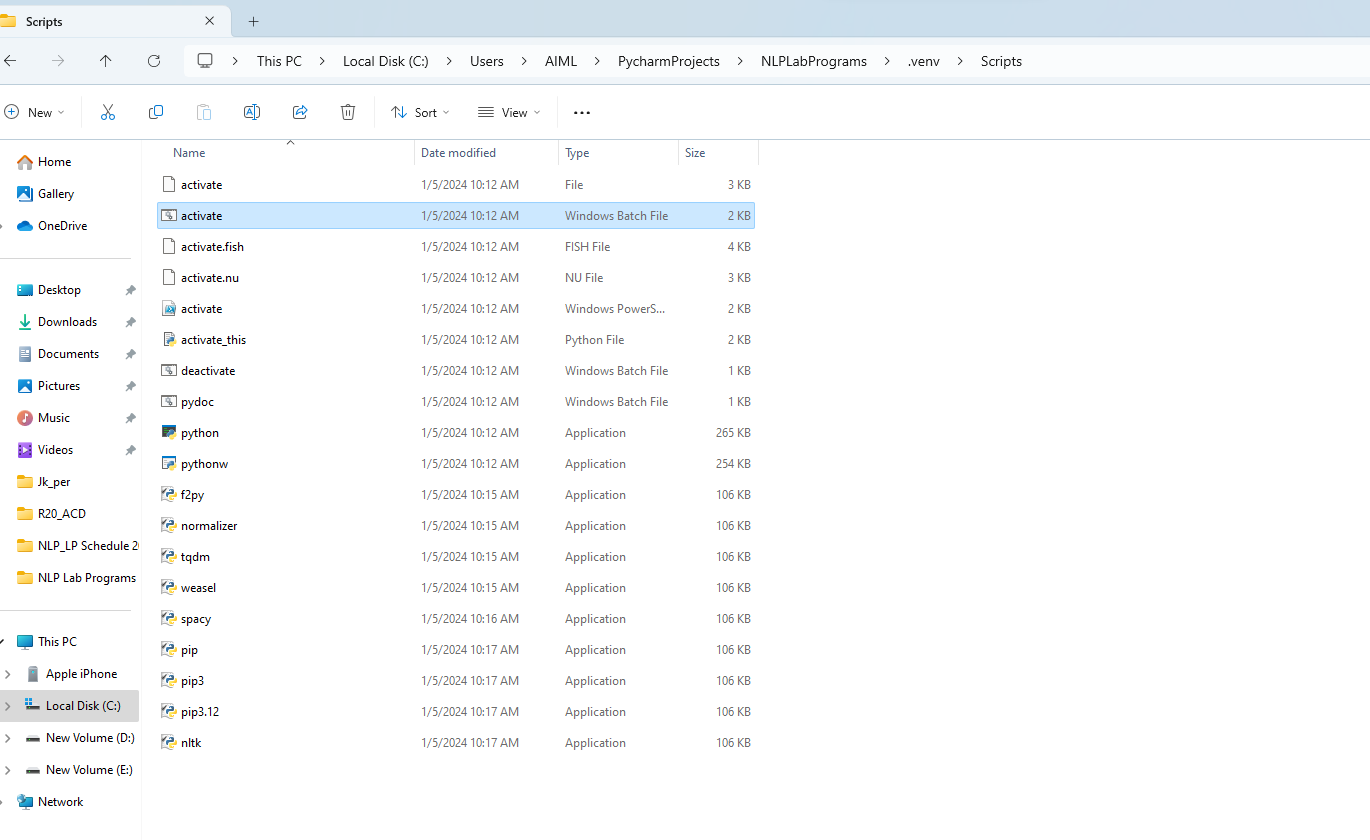
Run Command : pip show spacy



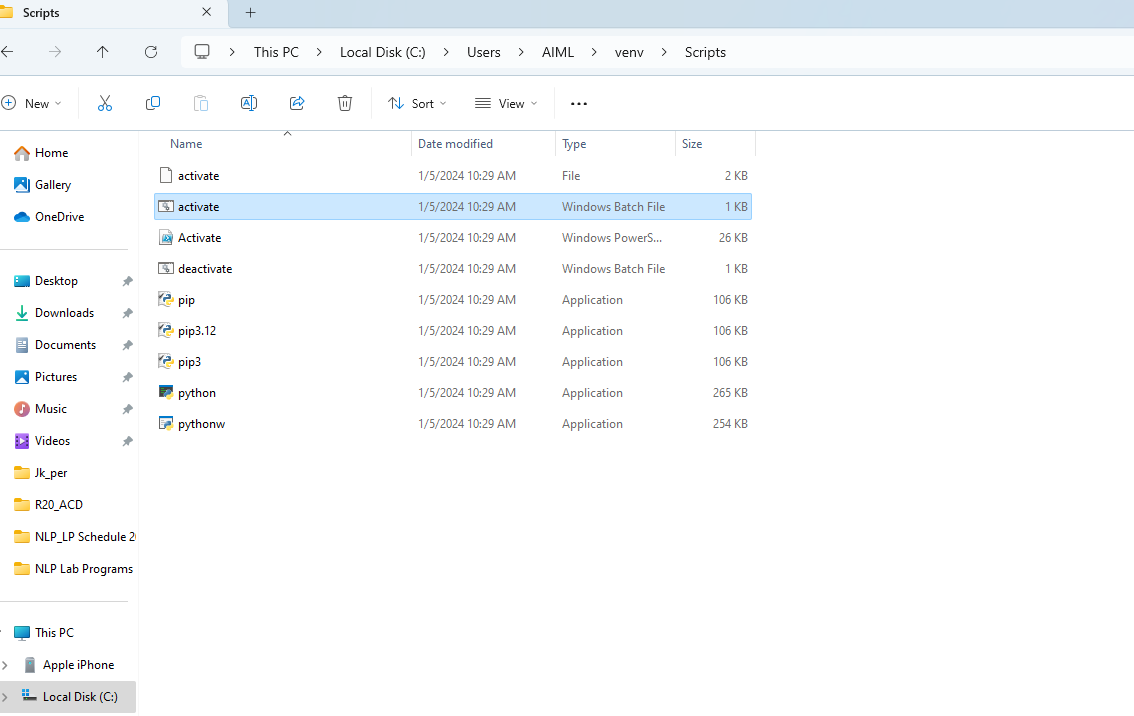
Virtual Environment : python -m venv venv



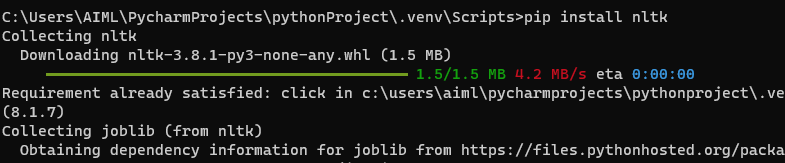
Double click on batch file “activate”



C:\Users\AIML\venv\Scripts



Run Command : pip install nltk

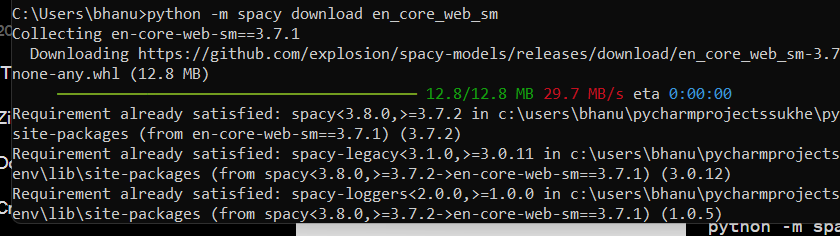


Run Command : pip install --upgrade numpy



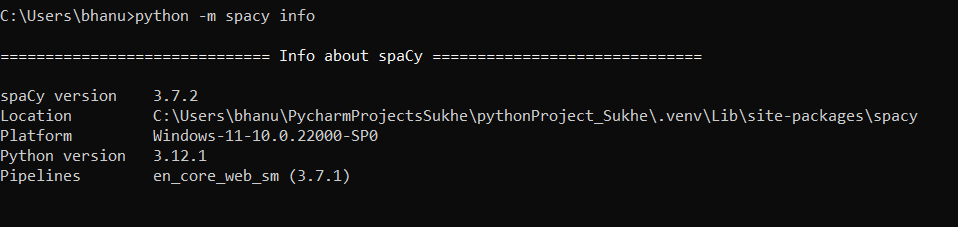
spaCy requires language models to perform NLP tasks. Download a language model using the following command:

**python -m spacy download en\_core\_web\_sm**



To verify that spaCy is installed correctly, you can run a simple test:

python -m spacy info



**RESULT :**

Hence, installation has been successfully implemented.

**2. AIM : IMPLEMENTATION OF RESUME SCREENING WITH PYTHON**

**DESCRIPTION :**

Artificial intelligence, along with text mining and natural language processing algorithms, can be applied for the development of programs (i.e. Applicant Tracking Systems) capable of screening objectively thousands of resumes in few minutes without bias to identify the best fit for a job opening based on thresholds, specific criteria or scores.These programs usually look for specific keywords; they sort resumes and rank them to determine the job applications that should be further reviewed by recruiters. While each company might have its own resume screening system, it is crucial for candidates to know how they work with the aim of improving their keywords selection based on the job opening they are applying to.

**CODING 1 :**

import nltk

import re

# Download NLTK data (if not already downloaded)

nltk.download('punkt')

nltk.download('stopwords')

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

def preprocess\_text(text):

# Tokenize into words

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

# Remove stop words and non-alphabetic words

stop\_words = set(stopwords.words('english'))

filtered\_words = [word for word in words if word.isalpha() and word not in stop\_words]

return filtered\_words

def calculate\_similarity(text1, text2):

# Calculate Jaccard Similarity

set1 = set(text1)

set2 = set(text2)

intersection = set1.intersection(set2)

union = set1.union(set2)

similarity = len(intersection) / len(union)

return similarity

def screen\_resume(job\_description, resume):

# Preprocess job description and resume

job\_desc\_tokens = preprocess\_text(job\_description)

resume\_tokens = preprocess\_text(resume)

# Calculate similarity

similarity = calculate\_similarity(job\_desc\_tokens, resume\_tokens)

return similarity

# Example job description and resume

job\_description = "Seeking a software engineer with experience in Python and machine learning & java."

resume = "Hi, I am Surekha and I am a software engineer with experience in Python, nlp and machine learning."

# Perform resume screening

similarity\_score = screen\_resume(job\_description, resume)

# Set a threshold for similarity (adjust as needed)

threshold = 0.5

# Print result

if similarity\_score >= threshold:

print("Congratulations!! Your resume matches the job description, we will schedule an Interview soon!!")

else:

print("Sorry!! Your resume does not match the job description, kindly acquire more knowldege & apply again, good luck!!")

**OUTPUT :**

C:\Users\AIML\PycharmProjects\NLPLabPrograms\.venv\Scripts\python.exe C:\Users\AIML\PycharmProjects\NLPLabPrograms\NltkSpacyAlternative.py

[nltk\_data] Downloading package punkt to

[nltk\_data] C:\Users\AIML\AppData\Roaming\nltk\_data...

[nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package stopwords to

[nltk\_data] C:\Users\AIML\AppData\Roaming\nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!

Congratulations!! Your resume matches the job description, we will schedule an Interview soon!!

Process finished with exit code 0

**CODING 2 :**

import nltk

import re

# Download NLTK data (if not already downloaded)

nltk.download('punkt')

nltk.download('stopwords')

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

def preprocess\_text(text):

# Tokenize into words

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

# Remove stop words and non-alphabetic words

stop\_words = set(stopwords.words('english'))

filtered\_words = [word for word in words if word.isalpha() and word not in stop\_words]

return filtered\_words

def calculate\_similarity(text1, text2):

# Calculate Jaccard Similarity

set1 = set(text1)

set2 = set(text2)

intersection = set1.intersection(set2)

union = set1.union(set2)

similarity = len(intersection) / len(union)

return similarity

def screen\_resume(job\_description, resume):

# Preprocess job description and resume

job\_desc\_tokens = preprocess\_text(job\_description)

resume\_tokens = preprocess\_text(resume)

# Calculate similarity

similarity = calculate\_similarity(job\_desc\_tokens, resume\_tokens)

return similarity

# Example job description and resume

job\_description = "Seeking a software engineer with experience in C and machine learning & java."

resume = "Hi, I am Surekha and I am a software engineer with experience in Python, nlp and machine learning."

# Perform resume screening

similarity\_score = screen\_resume(job\_description, resume)

# Set a threshold for similarity (adjust as needed)

threshold = 0.5

# Print result

if similarity\_score >= threshold:

print("Congratulations!! Your resume matches the job description, we will schedule an Interview soon!!")

else:

print("Sorry!! Your resume does not match the job description, kindly acquire more knowldege & apply again, good luck!!")

**OUTPUT :**

C:\Users\AIML\PycharmProjects\NLPLabPrograms\.venv\Scripts\python.exe C:\Users\AIML\PycharmProjects\NLPLabPrograms\NltkSpacyAlternative.py

[nltk\_data] Downloading package punkt to

[nltk\_data] C:\Users\AIML\AppData\Roaming\nltk\_data...

[nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package stopwords to

[nltk\_data] C:\Users\AIML\AppData\Roaming\nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!

Sorry!! Your resume does not match the job description, kindly acquire more knowldege & apply again, good luck!!

Process finished with exit code 0

**RESULT :**

Thus, we implemented of Resume Screening with Python successfully.

1. **AIM : CREATION OF NAMED ENTITY RECOGNITION WITH PYTHON.**

**DESCRIPTION :** Named Entity Recognition (NER) is a natural language processing (NLP) technique that involves identifying and classifying entities (such as names of persons, organizations, locations, dates, and other specific types of information) within a text. The goal of NER is to extract structured information from unstructured text data.In the context of NER, an "entity" refers to a specific named or classified object or concept, and the recognition part involves identifying the boundaries of these entities in a given text. NER is crucial for various NLP applications, including information retrieval, question answering, text summarization, and more.

**CODING :**

import re

def extract\_entities(text):

entities = []

# Define regular expressions for common entities (e.g., dates, numbers, names)

date\_pattern = r'\d{1,2}/\d{1,2}/\d{2,4}'

number\_pattern = r'\d+'

name\_pattern = r'[A-Z][a-z]+'

# Find dates

dates = re.findall(date\_pattern, text)

entities.extend([('DATE', date) for date in dates])

# Find numbers

numbers = re.findall(number\_pattern, text)

entities.extend([('NUMBER', number) for number in numbers])

# Find names

names = re.findall(name\_pattern, text)

entities.extend([('NAME', name) for name in names])

return entities

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage

text = "On 01/12/2024, John received 500 dollars from Mary."

entities = extract\_entities(text)

if entities:

print("Entities found:")

for entity\_type, entity\_value in entities:

print(f"{entity\_type}: {entity\_value}")

else:

print("No entities found.")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\NamedEntity.py

Entities found:

DATE: 01/12/2024

NUMBER: 01

NUMBER: 12

NUMBER: 2024

NUMBER: 500

NAME: On

NAME: John

NAME: Mary

Process finished with exit code 0

**RESULT :**

Thus, we implemented Named Entity Recognition with Python successfully.

1. **AIM : DEVELOPMENT OF SENTIMENT ANALYSIS WITH PYTHON.**

**DESCRIPTION :** To perform sentiment analysis in Python, you can use various libraries and approaches. One common and simple approach is to use the nltk library along with a pre-trained sentiment analysis model. In this example, the SentimentIntensityAnalyzer from nltk.sentiment is used to analyze sentiment. The polarity\_scores method returns a dictionary of sentiment scores, including a compound score that represents the overall sentiment.You can customize the sentiment categories (positive, negative, neutral) based on your specific requirements and the threshold values for the compound score.Keep in mind that this example is relatively basic.

**CODING 1 :**

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

# Download NLTK data (if not already downloaded)

nltk.download('vader\_lexicon')

def analyze\_sentiment(text):

# Initialize the sentiment intensity analyzer

sia = SentimentIntensityAnalyzer()

# Get sentiment scores

sentiment\_scores = sia.polarity\_scores(text)

# Determine sentiment based on the compound score

if sentiment\_scores['compound'] >= 0.05:

return 'Positive'

elif sentiment\_scores['compound'] <= -0.05:

return 'Negative'

else:

return 'Neutral'

# Example text

text = "I love working with Python! It's such a powerful language."

# text = "I am so stressed with Java as it is very difficult! It's such a difficult language."

# Perform sentiment analysis

result = analyze\_sentiment(text)

# Display the result

print(f"Sentiment: {result}")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\SentimentalAnalysis.py

[nltk\_data] Downloading package vader\_lexicon to

[nltk\_data] C:\Users\civilsys51\AppData\Roaming\nltk\_data...

[nltk\_data] Package vader\_lexicon is already up-to-date!

Sentiment: Positive

Process finished with exit code 0

**CODING 2 :**

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

# Download NLTK data (if not already downloaded)

nltk.download('vader\_lexicon')

def analyze\_sentiment(text):

# Initialize the sentiment intensity analyzer

sia = SentimentIntensityAnalyzer()

# Get sentiment scores

sentiment\_scores = sia.polarity\_scores(text)

# Determine sentiment based on the compound score

if sentiment\_scores['compound'] >= 0.05:

return 'Positive'

elif sentiment\_scores['compound'] <= -0.05:

return 'Negative'

else:

return 'Neutral'

# Example text

#text = "I love working with Python! It's such a powerful language."

text = "I am so stressed with Java as it is very difficult! It's such a difficult language."

# Perform sentiment analysis

result = analyze\_sentiment(text)

# Display the result

print(f"Sentiment: {result}")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\SentimentalAnalysis.py

[nltk\_data] Downloading package vader\_lexicon to

[nltk\_data] C:\Users\civilsys51\AppData\Roaming\nltk\_data...

[nltk\_data] Package vader\_lexicon is already up-to-date!

Sentiment: Negative

Process finished with exit code 0

**RESULT :**

Hence, we developed Sentiment Analysis with Python successfully.

1. **AIM : CREATE KEYWORD EXTRACTION WITH PYTHON.**

**DESCRIPTION :** Keyword extraction is the process of automatically identifying and extracting significant words or phrases from a piece of text. The objective is to distill the most relevant and important terms that represent the core topics or themes within the text. These extracted keywords can provide a concise summary or representation of the content, aiding in tasks such as document summarization, information retrieval, and content analysis.

**CODING :**

import re

from collections import Counter

def extract\_keywords(text, num\_keywords=5):

# Remove non-alphabetic characters and convert to lowercase

cleaned\_text = re.sub(r'[^a-zA-Z\s]', '', text.lower())

# Tokenize the text into words

words = cleaned\_text.split()

# Calculate word frequencies using Counter

word\_frequencies = Counter(words)

# Extract the top N keywords

top\_keywords = word\_frequencies.most\_common(num\_keywords)

return top\_keywords

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage

text = "Natural language processing is a subfield of artificial intelligence that focuses on the interaction between computers and humans using natural language."

keywords = extract\_keywords(text, num\_keywords=5)

if keywords:

print("Top Keywords:")

for keyword, count in keywords:

print(f"{keyword}: {count}")

else:

print("No keywords found.")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\KeywordExtraction.py

Top Keywords:

natural: 2

language: 2

processing: 1

is: 1

a: 1

Process finished with exit code 0

**RESULT :**

Thus, we created keyword extraction using Python successfully.

1. **AIM : IMPLEMENTATION OF SPELL CORRECTION MODEL WITH PYTHON.**

**DESCRIPTION :** A spell correction model is a computational system designed to automatically correct or suggest corrections for misspelled words in a given text. The primary goal of a spell correction model is to improve the accuracy and readability of textual data by identifying and fixing errors in spelling. The typical components and steps involved in a spell correction model include:

**1. Dictionary or Corpus:** A reference collection of correctly spelled words, often referred to as a dictionary or corpus, is used as a basis for comparison. This dataset helps the model identify whether a word is spelled correctly or not.

**2. Error Detection:** The model identifies potentially misspelled words in the input text. This can be done by comparing each word against the entries in the dictionary.

**3. Suggestion Generation:** For identified misspelled words, the model generates a list of candidate corrections. This list is usually based on various techniques, such as edit distance, where the model suggests words that are close in terms of character edits (insertions, deletions, substitutions).

**4. Ranking and Selection:** The generated suggestions are often ranked by likelihood or relevance. The model then selects the most probable correction as the suggested replacement for the misspelled word.

**5. Integration with Context:** Some spell correction models also consider the context of the surrounding words to improve accuracy. Context-aware models may use information from neighboring words to provide more relevant suggestions.

**6. User Feedback:** In interactive systems, users may be given the option to choose from a list of suggestions or manually input corrections. This feedback loop helps the model improve over time.

**CODING :**

import nltk

from nltk.corpus import words

from nltk.metrics.distance import edit\_distance

nltk.download('words')

class SpellCorrector:

def \_\_init\_\_(self):

self.word\_set = set(words.words())

def correct\_spelling(self, input\_word):

if input\_word.lower() in self.word\_set:

return input\_word # The word is correctly spelled

suggestions = self.get\_suggestions(input\_word)

if suggestions:

return min(suggestions, key=lambda x: edit\_distance(input\_word, x))

return input\_word # No suggestions found, return the original word

def get\_suggestions(self, input\_word):

# Generate suggestions based on edit distance

return [word for word in self.word\_set if edit\_distance(input\_word, word) <= 2]

if \_\_name\_\_ == "\_\_main\_\_":

spell\_corrector = SpellCorrector()

while True:

user\_input = input("Enter a word (type 'exit' to quit): ")

if user\_input.lower() == 'exit':

break

corrected\_word = spell\_corrector.correct\_spelling(user\_input)

print(f"Corrected word: {corrected\_word}")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\SpellCorrection.py

[nltk\_data] Downloading package words to

[nltk\_data] C:\Users\civilsys51\AppData\Roaming\nltk\_data...

[nltk\_data] Unzipping corpora\words.zip.

Enter a word (type 'exit' to quit): bt

eCorrected word: bit

Enter a word (type 'exit' to quit): exit

Process finished with exit code 0

**RESULT :**

Thus, we implement spell correction model using Python successfully.

1. **AIM : CREATION OF KEYBOARD AUTO CORRECTION MODEL.**

**DESCRIPTION :** A keyboard auto-correction model is a computational system designed to automatically correct or suggest corrections for words based on the proximity of keys on a keyboard. This type of correction model aims to address typing errors that occur when a user unintentionally types a word with characters adjacent to the intended characters on a standard keyboard layout.The key steps involved in the creation of a keyboard auto-correction model typically include:

**1. Keyboard Layout Definition:** The model is built based on a predefined keyboard layout, such as the QWERTY layout. It defines the proximity of keys for each character on the keyboard.

**2. Error Detection:** The model identifies words in the input text that are likely to contain typing errors. This can involve comparing each word against a reference dictionary or corpus.

**3. Suggestion Generation:** For identified words with potential errors, the model generates a list of candidate corrections. Suggestions are typically generated by considering words that are close in terms of keyboard proximity to the characters in the original word.

**4. Ranking and Selection:** The suggested corrections are often ranked by likelihood or relevance. The model then selects the most probable correction as the recommended replacement for the input word.

**5. Integration with Context:** Some models may consider the context of the surrounding words to improve the accuracy of corrections. Context-aware models use information from neighboring words to provide more relevant suggestions.

**CODING :**

import numpy as np

class KeyboardAutoCorrector:

def \_\_init\_\_(self, dictionary):

self.dictionary = set(word.lower() for word in dictionary)

def levenshtein\_distance(self, s1, s2):

if len(s1) < len(s2):

return self.levenshtein\_distance(s2, s1)

if len(s2) == 0:

return len(s1)

previous\_row = np.arange(len(s2) + 1)

for i, char1 in enumerate(s1):

current\_row = np.zeros(len(s2) + 1, dtype=int)

current\_row[0] = i + 1

for j, char2 in enumerate(s2):

insertions = previous\_row[j + 1] + 1

deletions = current\_row[j] + 1

substitutions = previous\_row[j] + (char1 != char2)

current\_row[j + 1] = min(insertions, deletions, substitutions)

previous\_row = current\_row

return previous\_row[-1]

def auto\_correct(self, word):

word\_lower = word.lower()

if word\_lower in self.dictionary:

return word # The word is already in the dictionary

corrected\_word = self.find\_closest\_word(word\_lower)

return corrected\_word

def find\_closest\_word(self, word):

min\_distance = float('inf')

closest\_word = word

for dictionary\_word in self.dictionary:

distance = self.levenshtein\_distance(word, dictionary\_word)

if distance < min\_distance:

min\_distance = distance

closest\_word = dictionary\_word

return closest\_word

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage with a small dictionary

word\_dictionary = ["python", "programming", "language", "spell", "correction"]

auto\_corrector = KeyboardAutoCorrector(word\_dictionary)

while True:

user\_input = input("Enter a word (type 'exit' to quit): ")

if user\_input.lower() == 'exit':

break

corrected\_word = auto\_corrector.auto\_correct(user\_input)

print(f"Corrected word: {corrected\_word}")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\AutoCorrectionModel.py

Enter a word (type 'exit' to quit): pyhton

Corrected word: python

Enter a word (type 'exit' to quit): programing

Corrected word: programming

Enter a word (type 'exit' to quit): EXIT

Process finished with exit code 0

**RESULT :**

Thus, we created keyboard autocorrection model successfully.

1. **AIM : DEVELOPMENT OF WHATSAPP GROUP CHAT ANALYSIS.**

**DESCTRIPTION :** Developing a WhatsApp group chat analysis involves creating a program or script to extract, process, and analyze the text data from a WhatsApp group chat. The goal is to gain insights into the dynamics, communication patterns, and other aspects of the group based on the messages exchanged. Here are the key steps involved in the development of a WhatsApp group chat analysis:

**1. Data Extraction:** Obtain the chat data from WhatsApp. You can export the chat as a text file or use WhatsApp's data export feature.

**2. Read Chat Data:** Develop a program to read the content of the exported chat file. This can involve reading a plain text file or parsing a specific format provided by WhatsApp.

**3. Message Extraction:** Extract individual messages from the chat data. Identify the sender, timestamp, and the content of each message. Regular expressions can be useful for this task.

**4. Participant Analysis:** Identify the participants in the group by analyzing the sender information in each message. Determine the number of messages sent by each participant.

**5. Message Frequency:** Analyze the frequency of messages sent by each participant. This can provide insights into the most active participants in the group.

**6. Word Frequency:** Analyze the frequency of words used in the chat. This can help identify common topics or frequently mentioned terms. Creating a word cloud is a visually appealing way to represent word frequency.

**7. Time Analysis:** Explore how the chat activity varies over time. This can involve analyzing messages per day, hour, or month to identify patterns.

**8. Visualizations:** Create visualizations such as bar charts, pie charts, or line graphs to represent various aspects of the group chat data.

**9. Word Sentiment Analysis (Optional):** If sentiment analysis is of interest, you can use external libraries or tools to determine the sentiment of the messages (positive, negative, neutral).

**10. Interactive Interface (Optional):** Develop an interactive interface or dashboard for users to explore the insights visually.

**INSTALLATION PROCEDURE :**

pip install matplotlib

pip install wordcloud

**CODING :**

import re

from collections import Counter

import matplotlib.pyplot as plt

from wordcloud import WordCloud

def read\_chat(file\_path):

with open(file\_path, 'r', encoding='utf-8') as file:

chat\_text = file.read()

return chat\_text

def extract\_messages(chat\_text):

# Define a regular expression pattern for extracting messages

pattern = re.compile(r'(\d{1,2}/\d{1,2}/\d{2,4},? \d{1,2}:\d{2}(?: [APMapm]{2})? - .+?: .+)')

# Extract messages using the pattern

messages = re.findall(pattern, chat\_text)

return messages

def analyze\_participants(messages):

participants = set()

for message in messages:

sender = re.search(r' - (.+?):', message)

if sender:

participants.add(sender.group(1))

return participants

def plot\_message\_frequency(messages):

senders = [re.search(r' - (.+?):', message).group(1) for message in messages if re.search(r' - (.+?):', message)]

counter = Counter(senders)

plt.bar(counter.keys(), counter.values())

plt.xlabel('Participants')

plt.ylabel('Number of Messages')

plt.title('Message Frequency by Participant')

plt.show()

def generate\_wordcloud(chat\_text):

wordcloud = WordCloud(width = 800, height = 800,

background\_color ='white',

stopwords = set(["media", "omitted"]),

min\_font\_size = 10).generate(chat\_text)

plt.figure(figsize = (8, 8), facecolor = None)

plt.imshow(wordcloud)

plt.axis("off")

plt.tight\_layout(pad = 0)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = "Chatfile.txt" #'path\_to\_your\_chat\_file.txt' # Replace with the path to your exported WhatsApp chat file

chat\_text = read\_chat(file\_path)

messages = extract\_messages(chat\_text)

participants = analyze\_participants(messages)

print("Participants:", participants)

plot\_message\_frequency(messages)

generate\_wordcloud(chat\_text)

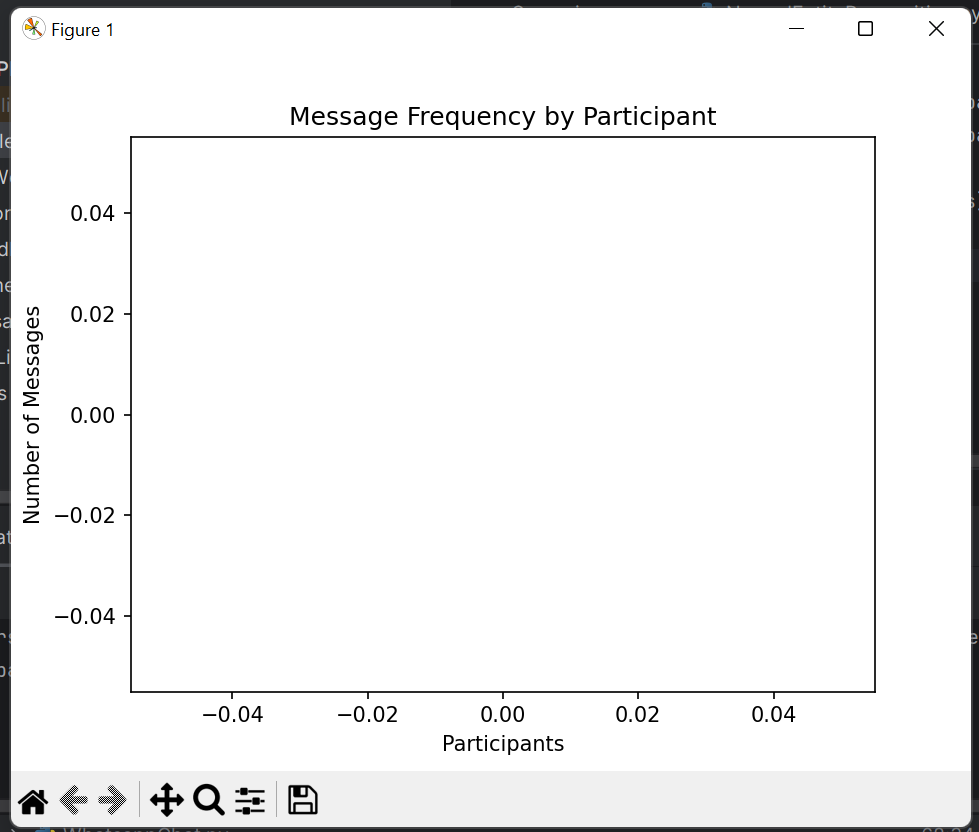
**Create new file with filename Chatfile.txt**

Hi how are you

**OUTPUT:**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\WhatsappGroup.py

Participants: set()





**RESULT:**

Thus, we implemented Whatsapp Group Analysis successfully.

1. **AIM : IMPLEMENTATION OF NEXT WORD PREDICTION MODEL**

**DESCRIPTION :** Implementing a next word prediction model involves training a language model to predict the next word in a sequence of words. In this example, I'll use a simple approach with the Markov chain model. This model predicts the next word based on the probability of the word given the preceding words.

**CODING :**

import random

from collections import defaultdict

class NextWordPredictor:

def \_\_init\_\_(self, n):

self.n = n # Order of the Markov chain

self.model = defaultdict(list)

def train\_model(self, text):

words = text.split()

for i in range(len(words) - self.n):

context = tuple(words[i:i + self.n])

next\_word = words[i + self.n]

self.model[context].append(next\_word)

def predict\_next\_word(self, context):

possible\_next\_words = self.model.get(context, [])

if possible\_next\_words:

return random.choice(possible\_next\_words)

else:

return None

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage

text\_data = "This is a simple example of a next word prediction model. Implementing this model can be educational and fun."

nwp\_model = NextWordPredictor(n=2) # Use a trigram model (n=2)

# Train the model

nwp\_model.train\_model(text\_data)

# Predict the next word given a context

context = tuple("word prediction".split())

predicted\_word = nwp\_model.predict\_next\_word(context)

if predicted\_word:

print(f"Predicted next word: {predicted\_word}")

else:

print("No prediction available.")

**OUTPUT :**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\NextWordPrediction.py

Predicted next word: model.

Process finished with exit code 0

**RESULT:**

Thus, we implemented Next Word Prediction Model Successfully.

1. **AIM : CREATION OF FAKE NEWS DETECTION MODEL**

**DESCRIPTION :** Creating a fake news detection model is a challenging task that typically involves natural language processing (NLP) and machine learning techniques. Here's a simplified example using Python and the scikit-learn library. This example uses a basic TF-IDF (Term Frequency-Inverse Document Frequency) approach and a Random Forest classifier. Please note that in a real-world scenario, you would need a much larger and more diverse dataset to train a robust model. Additionally, more advanced techniques, such as using pre-trained word embeddings (e.g., Word2Vec, GloVe) or deep learning models, might be necessary for better performance. Here are the general steps involved in creating a fake news detection model:

**1. Data Collection:** Gather a dataset of labeled news articles, indicating whether each article is real or fake.

**2. Data Preprocessing:** Clean and preprocess the text data, including tasks like removing stop words, stemming, and handling missing values.

**3. Feature Extraction:** Convert the text data into numerical features that can be used by a machine learning model. TF-IDF is a common technique for this purpose.

**4. Model Selection:** Choose a suitable machine learning algorithm for classification. Random Forest, Support Vector Machines (SVM), and neural networks are often used for this task.

**5. Model Training:** Train the selected model using the labeled training data.

**6. Model Evaluation:** Evaluate the model's performance on a separate test dataset. Common metrics include accuracy, precision, recall, and F1 score.

**7. Fine-Tuning:** Adjust hyperparameters and experiment with different feature extraction techniques or models to improve performance.

**INSTALLATION OF SOFTWARES :**

pip install pandas

pip install scikit-learn

**CODING :**

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

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

# Sample data for illustration

data = {

'text': ['Real news example', 'Fake news example', 'Another real news', 'Yet another fake news'],

'label': [1, 0, 1, 0] # 1 for real news, 0 for fake news

}

df = pd.DataFrame(data)

# Split the data into training and testing sets

train\_data, test\_data, train\_labels, test\_labels = train\_test\_split(df['text'], df['label'], test\_size=0.2, random\_state=42)

# Convert the text data into TF-IDF features

vectorizer = TfidfVectorizer()

train\_features = vectorizer.fit\_transform(train\_data)

test\_features = vectorizer.transform(test\_data)

# Train a Random Forest classifier

classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

classifier.fit(train\_features, train\_labels)

# Make predictions on the test set

predictions = classifier.predict(test\_features)

# Evaluate the model

accuracy = accuracy\_score(test\_labels, predictions)

report = classification\_report(test\_labels, predictions)

print(f"Accuracy: {accuracy}")

print("Classification Report:\n", report)

**OUTPUT:**

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\FakeNewsDetection.py

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Lib\site-packages\sklearn\metrics\\_classification.py:1471: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

Accuracy: 0.0

Classification Report:

precision recall f1-score support

0 0.00 0.00 0.00 1.0

1 0.00 0.00 0.00 0.0

accuracy 0.00 1.0

macro avg 0.00 0.00 0.00 1.0

weighted avg 0.00 0.00 0.00 1.0

Process finished with exit code 0

**RESULT:**

Hence, we created Fake News Detection Model Successfully.

1. **AIM :** **IMPLEMENTATION OF OCR USING TESSERACT**

**DESCRIPTION :** Optical Character Recognition (OCR) is a technology that extracts text from images or scanned documents. Tesseract is a popular open-source OCR engine. To use Tesseract in Python, you can use the pytesseract wrapper along with the Pillow library for image processing. Before starting, you need to install Tesseract on your system. You can download the installer from the official Tesseract GitHub page or use a package manager like apt or brew on Linux or macOS. After installing Tesseract, you also need to install the required Python libraries.

**SOFTWARES TO INSTALL :**

https://github.com/UB-Mannheim/tesseract/wiki

CLICK ON tesseract-ocr-w64-setup-5.3.3.20231005.exe (64 bit)

Download Software and Install

**COMMAND PROMPT INSTALL :**

pip install pytesseract

pip install Pillow

**CODING :**

from PIL import Image

import pytesseract

# Set the path to the Tesseract executable (update this based on your installation)

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def perform\_ocr(image\_path):

# Open the image using Pillow

image = Image.open(image\_path)

# Perform OCR using Tesseract

text = pytesseract.image\_to\_string(image)

return text

if \_\_name\_\_ == "\_\_main\_\_":

# Replace 'path/to/your/image.png' with the actual path to your image file

image\_path = r'C:\SUREKHA/News.png'

result = perform\_ocr(image\_path)

print("OCR Result:")

print(result)

**OUTPUT :**

Save below file with name News.png and give location path in code

image\_path = r'C:\SUREKHA/News.png'

****

C:\Users\civilsys51\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\civilsys51\PycharmProjects\pythonProject\Tesser.py

OCR Result:

\_DAILY \_NEWS\_

NEWS |

Process finished with exit code 0

**RESULT :**

Thus, we implemented OCR using Tesseract.