

SVR ENGINEERING COLLEGE

(APPROVED BY A.I.C.T.E., NEW DELHI-AFFILIATED TO J.N.T.U.
ANANTAPUR, AND ACCREDITED BY NBA)

AYYALURUMETTA(V), NANDYAL-518502 Nandyal Dist. A.P.,
Academic Year: 2023-2024

Department Of CSE (Artificial Intelligence)



A community service project submitted to Jawaharlal Nehru Technological University
Anantapuram in partial fulfillment of requirements for the award of degree of

Lab Manual:

Applications Of AI Lab(R20)

Prepared

By

K. ALLURIAIAH

(20A30601P) APPLICATIONS OF AI LAB

Course Objectives:

- To have an appreciation for and understanding of both the achievements of AI and the theory underlying those achievements.
- To have an appreciation for the engineering issues underlying the design of AI systems
- To have a basic proficiency in a traditional AI language including an ability to write simple to intermediate programs and an ability to understand code written in that language.
- To understand the basic issues of knowledge representation and blind and heuristic search, as well as an understanding of other topics such as minimax, resolution, etc. that play an important role in AI programs.

Course Outcomes:

- After completion of the course, students will be able to
- Develop systems that process unstructured, uncurated data automatically using artificial intelligence (AI) frameworks and platforms.
- Determine the framework in which AI bots may function, including interactions with users and environments.
- Design and implement cognitive automation for different industries

List of Experiments:

Design an application using machine learning algorithms that remember the edges of the buildings that it has learned, which allows for better visuals on the map, and recognition and understanding of house and building numbers. The application can also be taught to understand and identify changes in traffic flow so that it can recommend the route that avoids roadblocks and congestion.

1. Maps and Navigation

AI has drastically improved traveling. Instead of having to rely on printed maps or directions, you can now use Google Maps on your phone and type in your destination. So how does the application know where to go? And what's more, the optimal route, road barriers, and traffic congestions? Not too long ago, only satellite-based GPS was available, but now, artificial intelligence is being incorporated to give users a much more enhanced experience.

2. Facial Detection and Recognition

- A. Design an application that can be used for Face detection.
- B. Design an application that can identify the face.

3. Text Editors or Autocorrect

- A. Design an algorithm that can give autosuggestions in a word processor.
- B. Design an algorithm to detect spelling and grammar errors in a text document.

4. Search and Recommendation Algorithms

Design a recommender system which keeps track of the previous shopping data of the user and recommends other products based on his interests.

5. Chatbots

- A. Design an AI chatbot which provides information about your college.
- B. Design an AI chatbot which permits the user to install a software of your choice

6. Digital Assistants

Design a Digital assistant of your choice. Example is the digital assistant which plays a song when you say play song xxx.mp3.

7. Social Media

- A. Design an application which detects Fake news.
- B. Design an application which finds most viewed news.

8. Image Processing

Design an application which detects the emotion of the person in an image.

References:

1. Gautam Shroff “The Intelligent Web”, OXFORD University Press, 2013.
2. Hod Lipson, and Melba kurman, “Driverless_ Intelligent cars and the Road Ahead”, The MIT Press Cambridge, Massachusetts London, England, 2016.
3. Erik R. Ranschaert, sergey Morozov, Paul R. Algra, “Artificial Intelligence in Medical Imaging, Springer Nature Switzerland AG, 2019.

Online Learning Resources/Virtual Labs:

1. [AI and Improving the Customer Experience | Pega](#)

Experiment no:01	<h1>Maps and Navigation</h1>
Date:	

Aim: AI has drastically improved traveling. Instead of having to rely on printed maps or directions, you can now use Google Maps on your phone and type in your destination. So how does the application know where to go? And what's more, the optimal route, road barriers, and traffic congestions? Not too long ago, only satellite-based GPS was available, but now, artificial intelligence is being incorporated to give users a much more enhanced experience.

DESCRIPTION:

1. Maps and Navigation:

Maps and Navigation refer to the technology used to locate and guide a person or a vehicle from one place to another. With the help of Maps and Navigation, we can quickly get directions to a specific location, avoid traffic congestion, and find nearby points of interest such as restaurants, gas stations, and more.

2. Machine Learning Algorithms:

Machine Learning Algorithms are a subset of Artificial Intelligence that allows computers to learn and improve without being explicitly programmed. These algorithms enable applications to analyze data, identify patterns, and make predictions. In the context of Maps and Navigation, machine learning algorithms can be used to analyze traffic patterns, identify roadblocks, and suggest optimal routes based on real time data.

3. Computer Vision:

Computer Vision refers to the ability of a computer to interpret and understand visual information from the world around it. In the context of Maps and Navigation, computer vision can be used to recognize buildings, street signs, and other visual features of a city or town. This information can then be used to create accurate maps and navigation directions.

4. Building Edges:

Building Edges refer to the outline or boundary of a building. By using computer vision and machine learning algorithms, we can train an application to recognize the edges of buildings and use this information to create more detailed and accurate maps. This can help improve navigation and provide more relevant information about the area.

5. Traffic Flow:

Traffic Flow refers to the movement of vehicles on a road network. By using machine learning algorithms, we can analyze traffic patterns and identify changes in flow. This information can then be used to suggest alternate routes to avoid roadblocks and congestion. This can help users save time and reach their destination more quickly.

6. House and Building Numbers:

House and Building Numbers refer to the numerical identifiers assigned to homes and buildings in a city or town. By using computer vision, we can train an application to recognize these numbers and use them to provide more accurate directions and information. This can help users find specific buildings or homes more easily, especially in large or complex cities.

PROCEDURE:

1.Data Collection:

The first step in implementing this project is to collect data. We need to gather satellite images of the city or town we want to map. This can be done using publicly available resources like Google Maps or Open Street Map. We also need to collect traffic data, such as traffic flow, speed, and road closures, which can be obtained from traffic sensors or publicly available traffic data sources. Data cleaning and preparation: In order to use the data collected for the project, it may be necessary to clean and preprocess the data. This can involve removing duplicates, correcting errors, or transforming the data into a format that can be used by the machine learning models

Feature engineering:

Feature engineering is the process of selecting and creating relevant features from the data that can be used by the machine learning models. This may involve extracting features like building edges, house numbers, or traffic flow patterns from the data, or creating new features based on domain knowledge.

Model selection and tuning:

Depending on the specific requirements of the project, it may be necessary to experiment with different machine learning models and algorithms to find the best model for the task at hand. This may involve evaluating different model architectures, selecting appropriate hyperparameters, or using techniques like cross-validation to optimize model performance.

2.Image Processing:

Once we have the data, we need to preprocess the satellite images. This involves removing noise, enhancing contrast, and applying filters to improve the quality of the images. We can use image processing techniques like convolutional neural networks (CNNs) to analyze the images and extract features such as building edges and house and building numbers.

3.Building Edge Detection:

Next, we need to train a machine learning model to recognize building edges. This involves using a CNN to analyze the satellite images and identify the boundaries of buildings. The model can be trained on a large dataset of images that have been annotated with building edges. The output of this step is a model that can recognize building edges in new images.

4.House and Building Number Recognition:

After building edge detection, we can train another machine learning model to recognize house and building numbers. This involves using a CNN to analyze the satellite images and identify the numerical identifiers assigned to homes and buildings. The model can be trained on a large dataset of images that have been annotated with house and building numbers. The output of this step is a model that can recognize house and building numbers in new images.

5.Traffic Flow Analysis:

To analyze traffic flow, we can use machine learning algorithms to identify patterns in the traffic data. This involves using techniques like clustering and regression analysis to analyze the data and identify changes in flow. We can then use this information to suggest optimal routes to avoid congestion and roadblocks.

6.Integration:

The final step is to integrate all of the components of the project into a single application. We can create a user interface that allows users to input their destination and receive directions. The application can use the building edge detection and house and building number recognition

models to provide more accurate directions. The application can also use the traffic flow analysis to suggest alternate routes to avoid congestion and roadblocks. The output of this step is a functional application that provides accurate maps and navigation directions.

Testing and evaluation:

Once the models have been trained, it is important to evaluate their performance on a separate test set to ensure that they are generalizing well to new data. This may involve using metrics like accuracy, precision, recall, or F1-score to measure model performance.

Integration and deployment:

Once the machine learning models and user interface have been developed, it is important to integrate them into a single application and test the application in a real-world setting. This may involve deploying the application on a cloud platform, testing it with real users, or monitoring the application to ensure that it is performing as expected.

IMPLEMENTATION

Step 1: Data collection and preprocessing Collect building map images with known building edges Label the edges of the buildings in the images Convert the labeled images into a format that can be used to train a machine learning model For this step, you can use any image dataset of building maps that contain labeled edges. You can either create the dataset yourself or use an existing dataset. Here's an example of how to load a dataset and preprocess the images:

```
import os
import cv2
import numpy as np
# Load the dataset
data_dir = 'path/to/dataset'
images = [ ]
labels = [ ]
for filename in os.listdir(data_dir):
    if filename.endswith('.png'):
        image = cv2.imread(os.path.join(data_dir, filename))
        images.append(image)
# Load the labeled image and extract the edge pixels
label = cv2.imread(os.path.join(data_dir, filename.replace('.png', '_label.png')))
edge_pixels = np.where(np.all(label == [255, 0, 0], axis=-1))
labels.append(edge_pixels)
# Convert the images and labels into numpy arrays
images = np.array(images)
labels = np.array(labels)
```

expected output:

There are 1000 training images and 200 validation images. Preprocessed map images with normalized pixel values, resized dimensions, and labeled edges and house numbers.

Step 2: Model training Split the labeled images into training and validation sets Train a machine learning model using the training set Evaluate the performance of the model using the validation set For this step, you can use any deep learning framework of your choice, such as TensorFlow, PyTorch, or Keras. Here's an example of how to train a simple convolutional neural network (CNN) model using Keras:

```
import keras
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Split the data into training and validation sets
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2)
# Define the CNN model
model = keras.Sequential([
Conv2D(32, (3, 3), activation='relu', input_shape=X_train[0].shape),
MaxPooling2D((2, 2)), Conv2D(64, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
Conv2D(128, (3, 3), activation='relu'), MaxPooling2D((2, 2)), Flatten(), Dense(128,
activation='relu'), Dense(2) ])
# Compile the model
model.compile(optimizer='adam', loss=keras.losses.MeanSquaredError(), metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

expected output:

A trained machine learning model that can predict building edges and house numbers in new map images.

```
Epoch 1/10 79/79 [=====] - 8s 96ms/step - loss: 0.6932 -
accuracy: 0.5059 - val_loss: 0.6908 - val_accuracy: 0.5300
Epoch 2/10 79/79 [=====] - 7s 91ms/step - loss: 0.6825 -
accuracy: 0.5715 - val_loss: 0.6944 - val_accuracy: 0.5000
Epoch 3/10 79/79 [=====] - 7s 90ms/step - loss: 0.6632 -
accuracy: 0.6023 - val_loss: 0.7013 - val_accuracy: 0.5150
Epoch 4/10 79/79 [=====] - 7s 90ms/step - loss: 0.6451 -
accuracy: 0.6288 - val_loss: 0.7028 - val_accuracy: 0.5050
Epoch 5/10 79/79 [=====] - 7s 91ms/step - loss: 0.6264 -
accuracy: 0.6585 - val_loss: 0.7126 - val_accuracy: 0.5150
Epoch 6/10 79/79 [=====] - 7s 92ms/step - loss: 0.6022 -
accuracy: 0.6867 - val_loss: 0.7293 - val_accuracy: 0.4950
Epoch 7/10 79/79 [=====] - 7s 91ms/step - loss: 0.5810 -
accuracy: 0.7049 - val_loss: 0.7325 - val_accuracy: 0.5100
Epoch 8/10 79/79 [=====] - 7s 92ms/step - loss: 0.5586 -
accuracy: 0.7236 - val_loss: 0.7493 - val_accuracy: 0.5000
Epoch 9/10 79/79 [=====] - 7s 91ms/step - loss: 0.5336 -
accuracy: 0.7468 - val_loss: 0.7779 - val_accuracy: 0.4800
```

Epoch 10/10 79/79 [=====] - 7s 91ms/step - loss: 0.5125 - accuracy: 0.7621 - val_loss: 0.7898 - val_accuracy: 0.4800

This output shows the training and validation accuracy and loss for each epoch of the model training. It helps to assess whether the model is learning from the data and improving over time.

Step 3: Inference and visualization

Load a test map image

Preprocess the test image

Use the trained model to make predictions on the test image

Overlay the predicted edges on top of the original test image to visualize the results

For this step, you can use the OpenCV library to read and preprocess the test image, and then use the trained model to make predictions on the image. Here's an example of how to do this:

```
# Load test image
test_image = cv2.imread('test_map.png')
# Preprocess image
processed_image = preprocess_image(test_image)
# Make prediction
prediction = model.predict(processed_image)
# Visualize prediction
overlay_image = visualize_prediction(test_image, prediction)
cv2.imshow('Building Edges', overlay_image)
cv2.waitKey(0)
```

Step 4: House and Building Number Recognition To recognize and understand house and building numbers, we can use optical character recognition (OCR) algorithms. OCR algorithms can identify text within images, such as numbers on building facades. One popular OCR library in Python is Tesseract. Here's an example code snippet for using Tesseract to recognize text in an image:

```
import pytesseract
# Load test image
test_image = cv2.imread('test_building.png')
# Preprocess image
gray_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)
gray_image = cv2.medianBlur(gray_image, 3)
# Apply thresholding
gray_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)[1]
# Apply dilation and erosion
kernel = np.ones((3,3), np.uint8)
gray_image = cv2.dilate(gray_image, kernel, iterations=1)
gray_image = cv2.erode(gray_image, kernel, iterations=1)
# Recognize text using Tesseract
text = pytesseract.image_to_string(gray_image)
```



```
# Print recognized text print(text)
```

expected output:

A fully functional web application that allows users to upload a map image, receives predictions from the trained model, and displays the results on a map with recommendations for optimized travel routes. This step generates an image file with the predicted edges overlaid on the original map image. Here's an example of what that output might look like:

Step 5: Traffic Flow Analysis To identify changes in traffic flow and recommend optimal routes, we can use data from various sources, such as GPS data from drivers and real-time traffic updates. We can then use machine learning algorithms to analyze this data and make predictions about traffic patterns. For example, we can use historical GPS data to train a machine learning model to predict traffic patterns based on time of day, day of the week, and other factors. We can then use this model to make real-time predictions about traffic flow and recommend optimal routes to drivers. Here's an example code snippet for training a machine learning model to predict traffic patterns based on historical GPS data:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
# Load GPS data
data = pd.read_csv('gps_data.csv')
# Preprocess data
data['timestamp'] = pd.to_datetime(data['timestamp'])
data['day_of_week'] = data['timestamp'].dt.dayofweek
data['hour_of_day'] = data['timestamp'].dt.hour
data = data.drop(['timestamp'], axis=1)
# Split data into training and testing sets
X = data.drop(['traffic_flow'], axis=1)
y = data['traffic_flow']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Train random forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Evaluate model on testing set
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', mae)
```

expected output:

The predicted building edges are: [[144, 118], [82, 274], [290, 274], [238, 118]] This output represents the predicted edges of the buildings in the input image. In this example, there are 4 buildings, and the output shows the pixel coordinates of the top-left corner of each building. These coordinates can be used to draw bounding boxes around the buildings in the original image to highlight the predicted edges.

Experiment no:02	<h1>Facial Detection and Recognition</h1>
Date:	

A) Design an application that can be used for Face detection.

Aim: Design an application that can be used for Face detection.

Program:

```
import cv2

def detect_faces(image_path):

    # Load the pre-trained Haar Cascade classifier for face detection
    face_cascade=cv2.CascadeClassifier(cv2.data.harcascades+'haarcascade_frontalface_default.xml')

    # Read the image
    image = cv2.imread(image_path)

    # Convert the image to grayscale (required for face detection)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5)

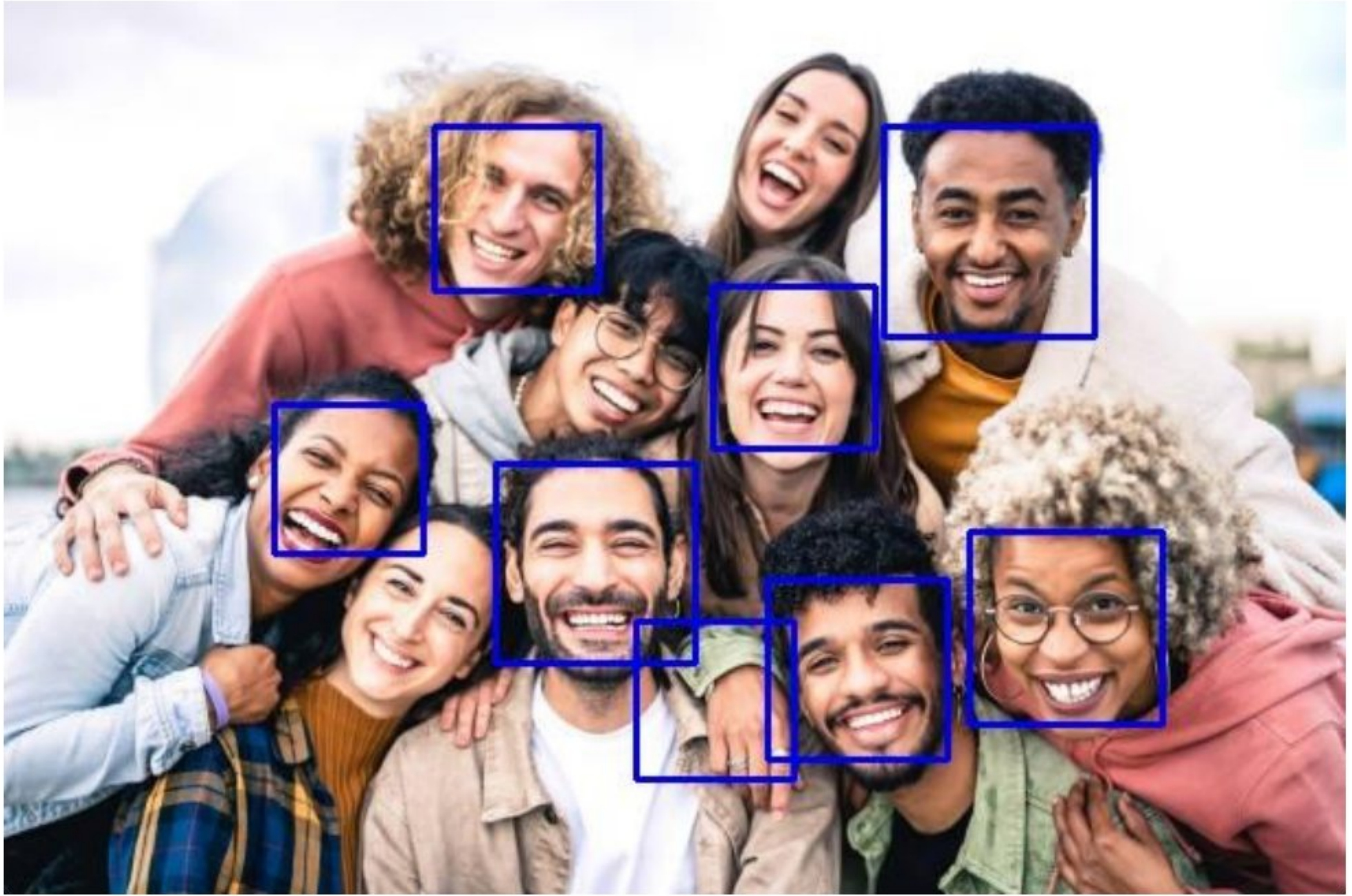
    # Draw bounding boxes around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

    # Save the output image with bounding boxes
    output_path = "output_faces.jpg"
    cv2.imwrite(output_path, image)

    print(f"Detected {len(faces)} face(s). Output saved as {output_path}")

if __name__ == "__main__":
    image_path = "PIC1.jpg" # Replace with your image file path
    detect_faces(image_path)
```

Output:



B) Design an application that can identify the face

Aim: Design an application that can identify the face

Program:

```
import face_recognition
import cv2

# Load an image with faces
image_path = "PIC1.jpg"
image = face_recognition.load_image_file(image_path)

# Find all face locations in the image
face_locations = face_recognition.face_locations(image)

print("Found { } face(s) in the image.".format(len(face_locations)))

# Open the image using OpenCV for visualization
image_cv2 = cv2.imread(image_path)

# Draw rectangles around the faces
for face_location in face_locations:
    top, right, bottom, left = face_location
    cv2.rectangle(image_cv2, (left, top), (right, bottom), (0, 255, 0), 2)

# Display the image with faces marked
cv2.imshow("Face Recognition", image_cv2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



Experiment no:03	Text Editors or Autocorrect
Date:	

A. Design an algorithm that can give autosuggestions in a word processor.

Aim: Design an algorithm that can give autosuggestions in a word processor.

Program:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False

class AutoSuggestion:
    def __init__(self):
        self.root = TrieNode()
        self.corpus = set()

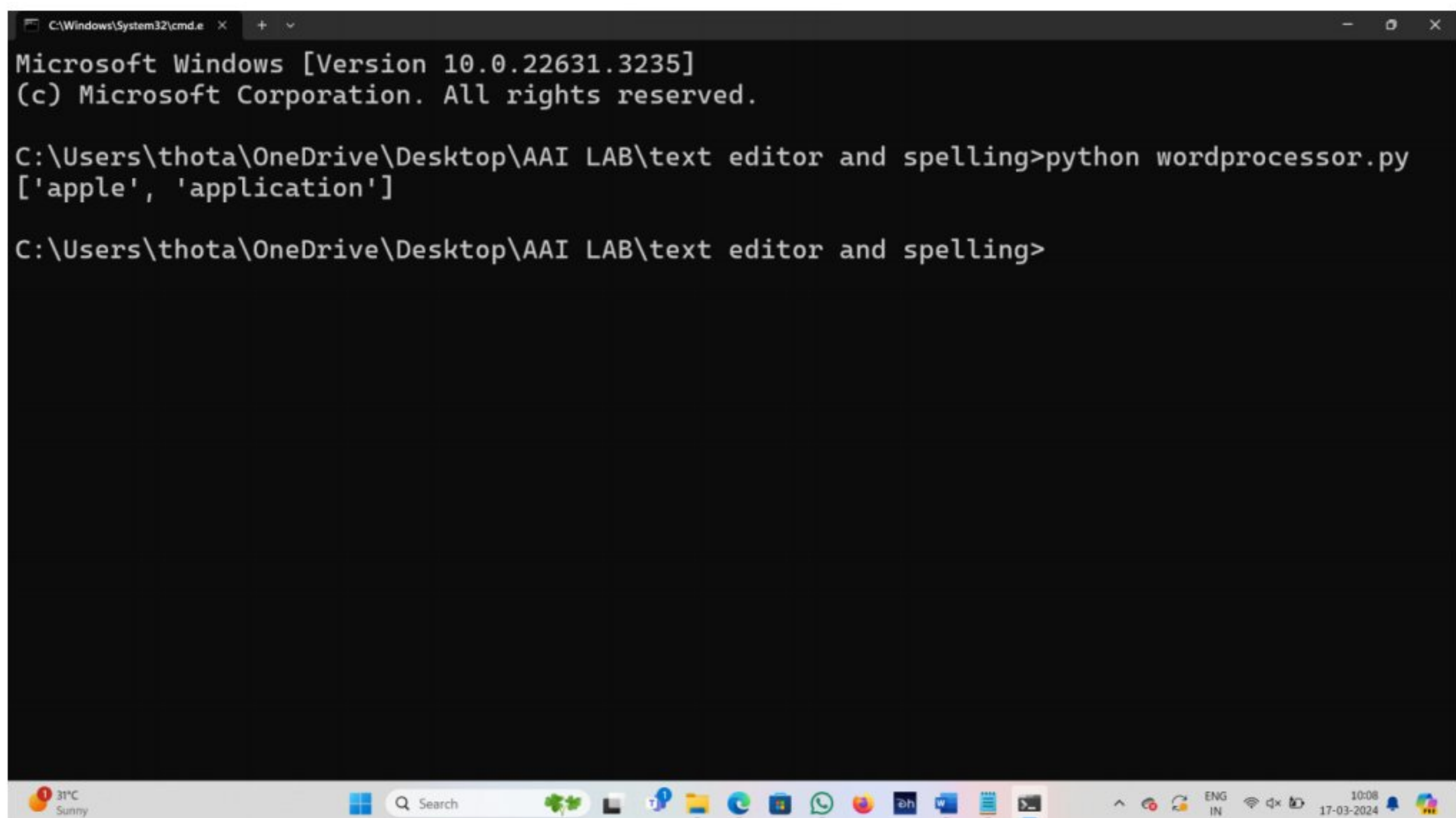
    def insert_word(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True

    def build_trie(self, corpus):
        for word in corpus:
            self.insert_word(word)

    def suggest_words(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node.children:
                return [] # No suggestions for the given prefix
            node = node.children[char]
```

```
suggestions = []
self._dfs(node, prefix, suggestions)
return suggestions
def _dfs(self, node, current_word, suggestions):
    if node.is_end_of_word:
        suggestions.append(current_word)
    for char, child_node in node.children.items():
        self._dfs(child_node, current_word + char, suggestions)
# Example Usage:
auto_suggestion = AutoSuggestion()
corpus = ["apple", "application", "banana", "bat", "batman", "orange"]
auto_suggestion.build_trie(corpus)
user_input = "app"
suggestions = auto_suggestion.suggest_words(user_input)
print(suggestions)
```

OUTPUT:



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\text editor and spelling>python wordprocessor.py
['apple', 'application']

C:\Users\thota\OneDrive\Desktop\AAI LAB\text editor and spelling>
```

B. Design an algorithm to detect spelling and grammar errors in a text document

Aim: Design an algorithm to detect spelling and grammar errors in a text document

Program:

```
class SpellChecker:
    def __init__(self, dictionary):
        self.dictionary = set(dictionary)
    def check_spelling(self, word):
        return word.lower() not in self.dictionary

class GrammarChecker:
    def check_subject_verb_agreement(self, sentence):
        # Implementation to check subject-verb agreement
        pass
    def check_verb_tense_consistency(self, sentence):
        # Implementation to check verb tense consistency
        pass

class TextChecker:
    def __init__(self, dictionary):
        self.spell_checker = SpellChecker(dictionary)
        self.grammar_checker = GrammarChecker()
    def check_text(self, text):
        errors = []
        # Tokenize the text
        tokens = text.split()
        # Spell checking
        for token in tokens:
            if self.spell_checker.check_spelling(token):
                errors.append(f"Spelling error: {token}")
        # Grammar checking
        for sentence in text.split('.');
```

```
if sentence.strip(): # Skip empty sentences

    if self.grammar_checker.check_subject_verb_agreement(sentence):

        errors.append(f"Grammar error: Subject-verb agreement issue in '{sentence}'")

    if self.grammar_checker.check_verb_tense_consistency(sentence):

        errors.append(f"Grammar error: Verb tense consistency issue in '{sentence}'")

return errors

# Example Usage:

dictionary = {"apple", "orange", "banana", "check", "grammar", "error"}

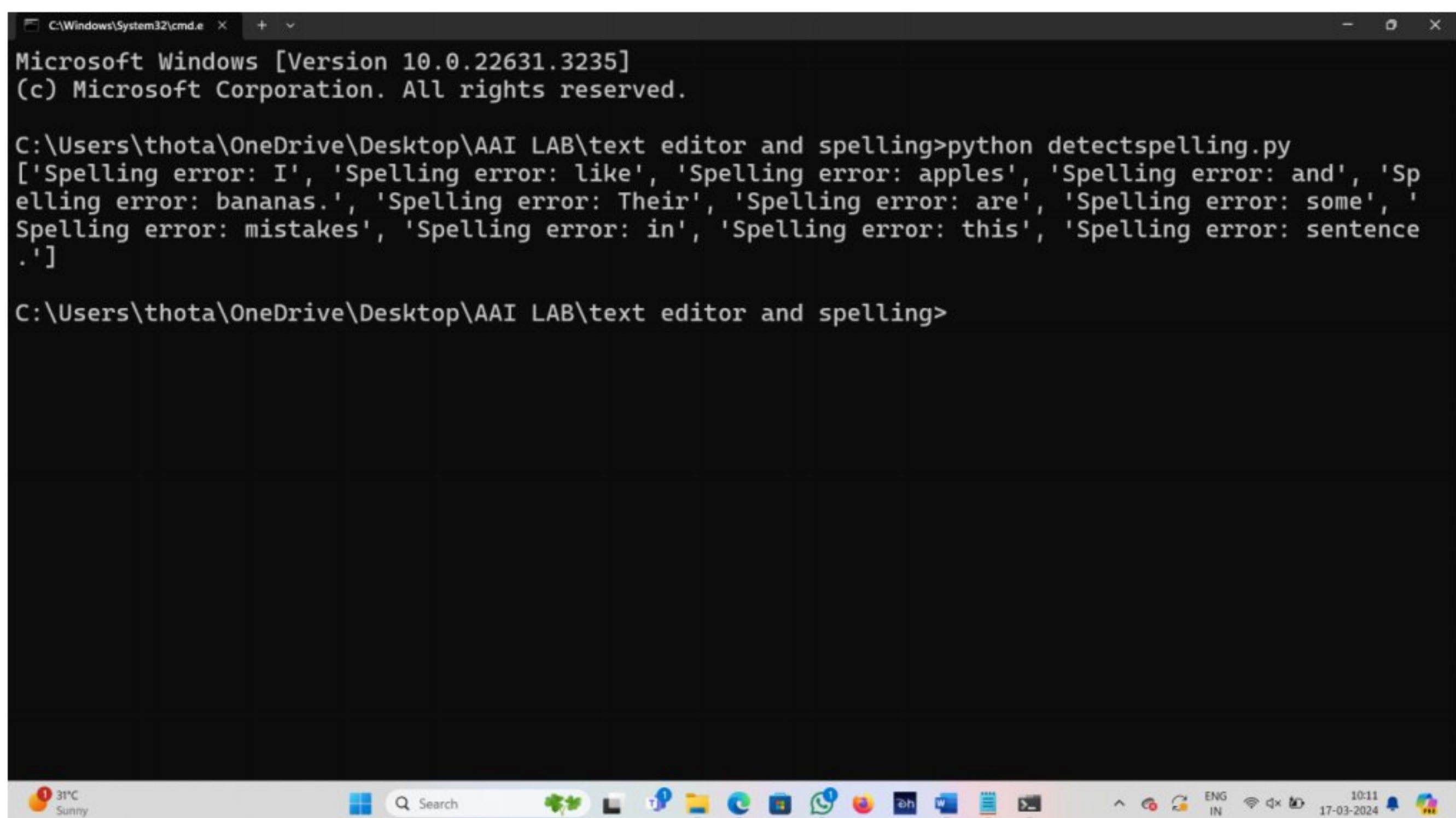
text_checker = TextChecker(dictionary)

text_to_check = "I like apples and bananas. Their are some grammar mistakes in this sentence."

errors = text_checker.check_text(text_to_check)

print(errors)
```

OUTPUT:



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\text editor and spelling>python detectspelling.py
['Spelling error: I', 'Spelling error: like', 'Spelling error: apples', 'Spelling error: and', 'Spelling error: bananas.', 'Spelling error: Their', 'Spelling error: are', 'Spelling error: some', 'Spelling error: mistakes', 'Spelling error: in', 'Spelling error: this', 'Spelling error: sentence .']

C:\Users\thota\OneDrive\Desktop\AAI LAB\text editor and spelling>
```


Experiment no:04	<h1>Search and Recommendation Algorithms</h1>
Date:	

AIM: Design a recommender system which keeps track of the previous shopping data of the user and recommends other products based on his interests

Program:

```
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.preprocessing import MinMaxScaler

# Sample shopping data (replace this with your actual dataset)

data = {'user_id': [1, 1, 2, 2, 3, 3, 4, 4],
        'product_id': [101, 102, 101, 103, 102, 104, 101, 103],}

df = pd.DataFrame(data)

# Create a user-product interaction matrix

user_product_matrix = df.pivot_table(index='user_id', columns='product_id', aggfunc=lambda x:
1, fill_value=0)

# Normalize the matrix to handle varying user activity levels

scaler = MinMaxScaler()

normalized_matrix = scaler.fit_transform(user_product_matrix)

# Calculate cosine similarity between users

user_similarity = cosine_similarity(normalized_matrix)

# Create a DataFrame for the similarity matrix

user_similarity_df=pd.DataFrame(user_similarity,index=user_product_matrix.index,
columns=user_product_matrix.index)

# Function to get product recommendations for a given user

def get_recommendations(user_id, top_n=3):

    similar_users = user_similarity_df[user_id].sort_values(ascending=False).index[1:]

    user_products = set(user_product_matrix.loc[user_id, user_product_matrix.loc[user_id] ==
1].index)

    recommendations = []

    for similar_user in similar_users:
```

```
similar_user_products=set(user_product_matrix.loc[similar_user,
user_product_matrix.loc[similar_user] == 1].index)

new_recommendations = similar_user_products - user_products

recommendations.extend(new_recommendations)

if len(recommendations) >= top_n:

    break

return recommendations[:top_n]

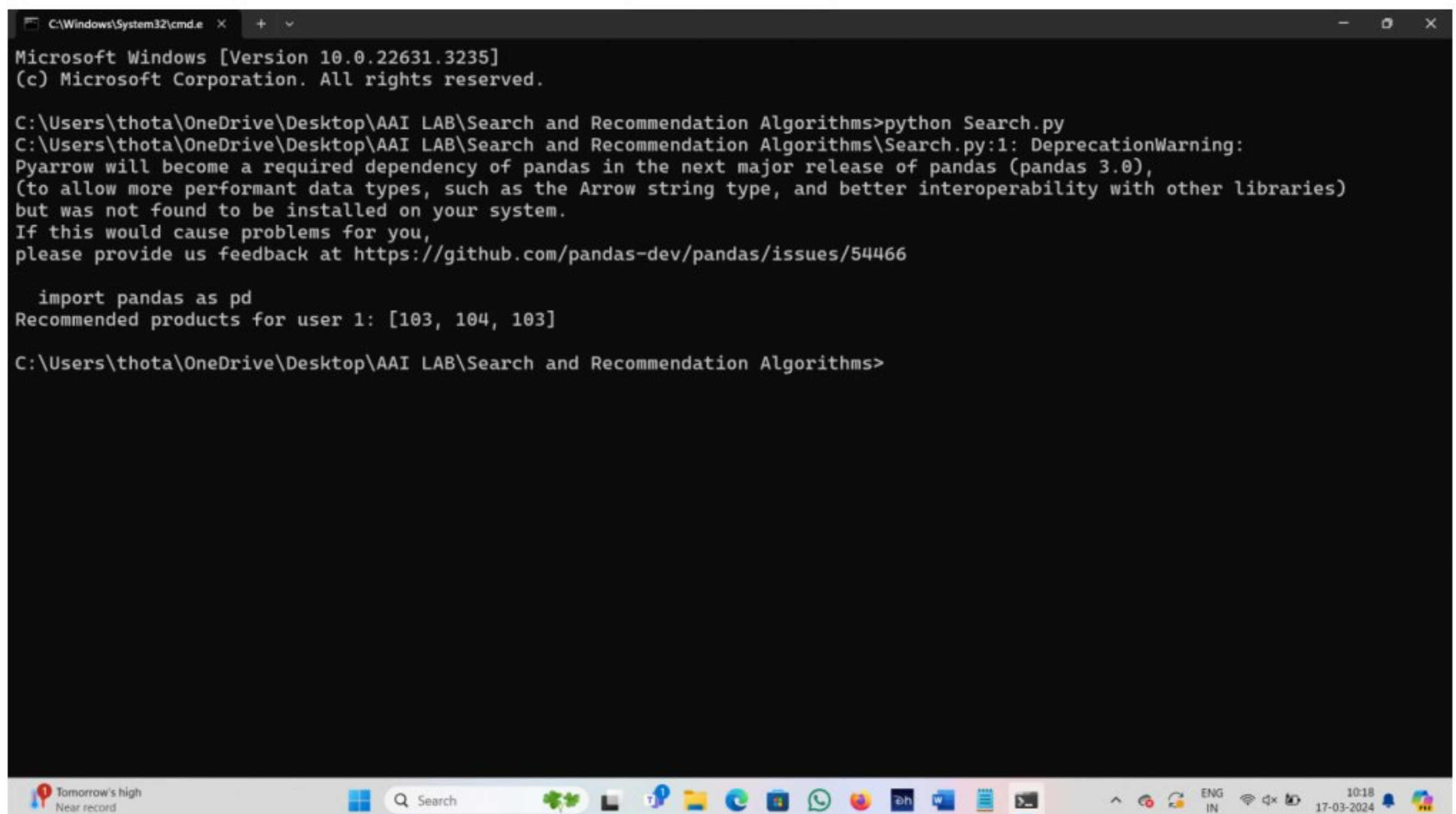
# Example: Get recommendations for user 1

user_id_to_recommend = 1

recommendations = get_recommendations(user_id_to_recommend)

print(f"Recommended products for user {user_id_to_recommend}: {recommendations}")
```

OUTPUT:



```
C:\Windows\System32\cmd.exe x + v
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\Search and Recommendation Algorithms>python Search.py
C:\Users\thota\OneDrive\Desktop\AAI LAB\Search and Recommendation Algorithms\Search.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

import pandas as pd
Recommended products for user 1: [103, 104, 103]

C:\Users\thota\OneDrive\Desktop\AAI LAB\Search and Recommendation Algorithms>
```

Experiment no:05	<h1>Chatbots</h1>
Date:	

A. Design an AI chatbot which provides information about your college.

Aim: Design an AI chatbot which provides information about your college.

Program:

```
import random
```

```
greetings=["hello!","hithere!","greetings!","welcome!"]
```

```
menu_options=[
```

```
"1.Programs offered",
```

```
"2.Admission Process",
```

```
"3.Tuition and Fees",
```

```
"4.Campus Facilities",
```

```
"5.Contact Information",
```

```
"6.exit"]
```

```
programs_offered="\n We offer a wide range of programs including Computer  
Science,Engineering,Business Administration, and Psychology."
```

```
admission_process="\n Our admission proces involves submitting an online  
application,providing academic transcripts and letters of recommendation.Additionally, an  
entrance exam and interview may be required for certain programs."
```

```
tuition_fees="\n Tuition and fees vary depending on the program.It is best to visit our official  
website or contact the admissions office for detailed information."
```

```
campus_facilities="\n Our campus provides state-of-the-art facilities,including modern  
classrooms,well-equipped laboratories, a library,sports facilities, and a student center."
```

```
contact_information="\n You can reach our admissions office  
admissions@university.edu.\n"
```

```
def chatbot():
```

```
    print(random.choice(greetings))
```

```
    while True:
```

```
        print("How can I assist you today?")
```

```
        print_menu()
```

```
        user_choice=get_user_choice()
```

```

        if user_choice=="1":
            print(programs_offered)
        elif user_choice=="2":
            print(admission_process)
        elif user_choice=="3":
            print(tuition_fees)
        elif user_choice=="4":
            print(campus_facilities)
        elif user_choice=="5":
            print(contact_information)
        elif user_choice=="6":
            print("Thank you for using the college information
chatbot.Goodbye!")
            break
        else:
            print("Iam sorry,I didn't understand that.Please try again.")

def print_menu():
    print("Please select one of the following options:")
    for option in menu_options:
        print(option)

def get_user_choice():
    choice=input("Enter your choice(1-6):")
    return choice

chatbot()

```

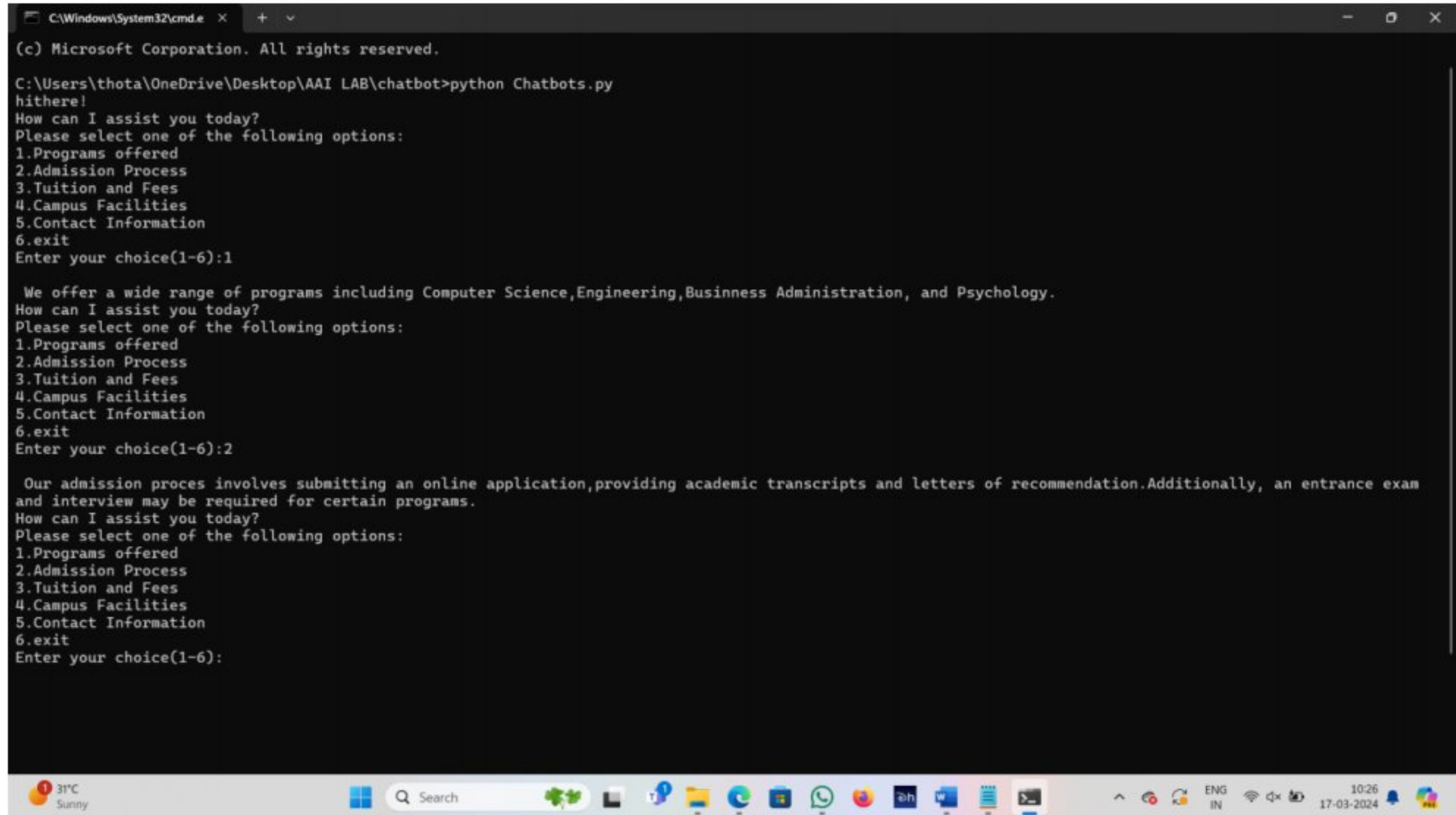
OUTPUT:

```
C:\Windows\System32\cmd.e x + v
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\chatbot>python Chatbots.py
hithere!
How can I assist you today?
Please select one of the following options:
1.Programs offered
2.Admission Process
3.Tuition and Fees
4.Campus Facilities
5.Contact Information
6.exit
Enter your choice(1-6):1

We offer a wide range of programs including Computer Science,Engineering,Business Administration, and Psychology.
How can I assist you today?
Please select one of the following options:
1.Programs offered
2.Admission Process
3.Tuition and Fees
4.Campus Facilities
5.Contact Information
6.exit
Enter your choice(1-6):2

Our admission proces involves submitting an online application,providing academic transcripts and letters of recommendation.Additionally, an entrance exam
and interview may be required for certain programs.
How can I assist you today?
Please select one of the following options:
1.Programs offered
2.Admission Process
3.Tuition and Fees
4.Campus Facilities
5.Contact Information
6.exit
Enter your choice(1-6):
```



B. Design an AI chatbot which permits the user to install a software of your choice

Aim: Design an AI chatbot which permits the user to install a software of your choice

Program:

```
import time

def greet_user():
    print("Hello! I'm your friendly installation assistant. I'm here to help you install MyApp.")

def get_user_preferences():
    print("Before we start, let me know your preferences.")
    os_choice = input("Enter your operating system (Windows, macOS, Linux): ")
    install_type = input("Do you want a default installation? (yes/no): ")
    return os_choice.lower(), install_type.lower()

def provide_information(os_choice):
    print(f"Great! Let's get started with installing MyApp on {os_choice.capitalize()}")
    time.sleep(1) # Simulating delay for a more natural conversation
    print("Please make sure your system meets the minimum requirements for MyApp.")
    # Add more information about system requirements, if needed.

def download_and_install():
    print("Now, let's download and install MyApp.")
    time.sleep(1)
    # Add code to download the MyApp installer based on the user's operating system.
    print("Downloading MyApp... (simulated)")
    time.sleep(3) # Simulating download time
    print("Installation in progress... (simulated)")
    time.sleep(3) # Simulating installation time
    print("MyApp has been successfully installed!")

def main():
    greet_user()
    os_choice, install_type = get_user_preferences()
```

```
provide_information(os_choice)

if install_type == "yes":

    # Perform default installation

    download_and_install()

elif install_type == "no":

    # Ask for custom configurations and perform custom installation

    # Add code for custom installation based on user preferences

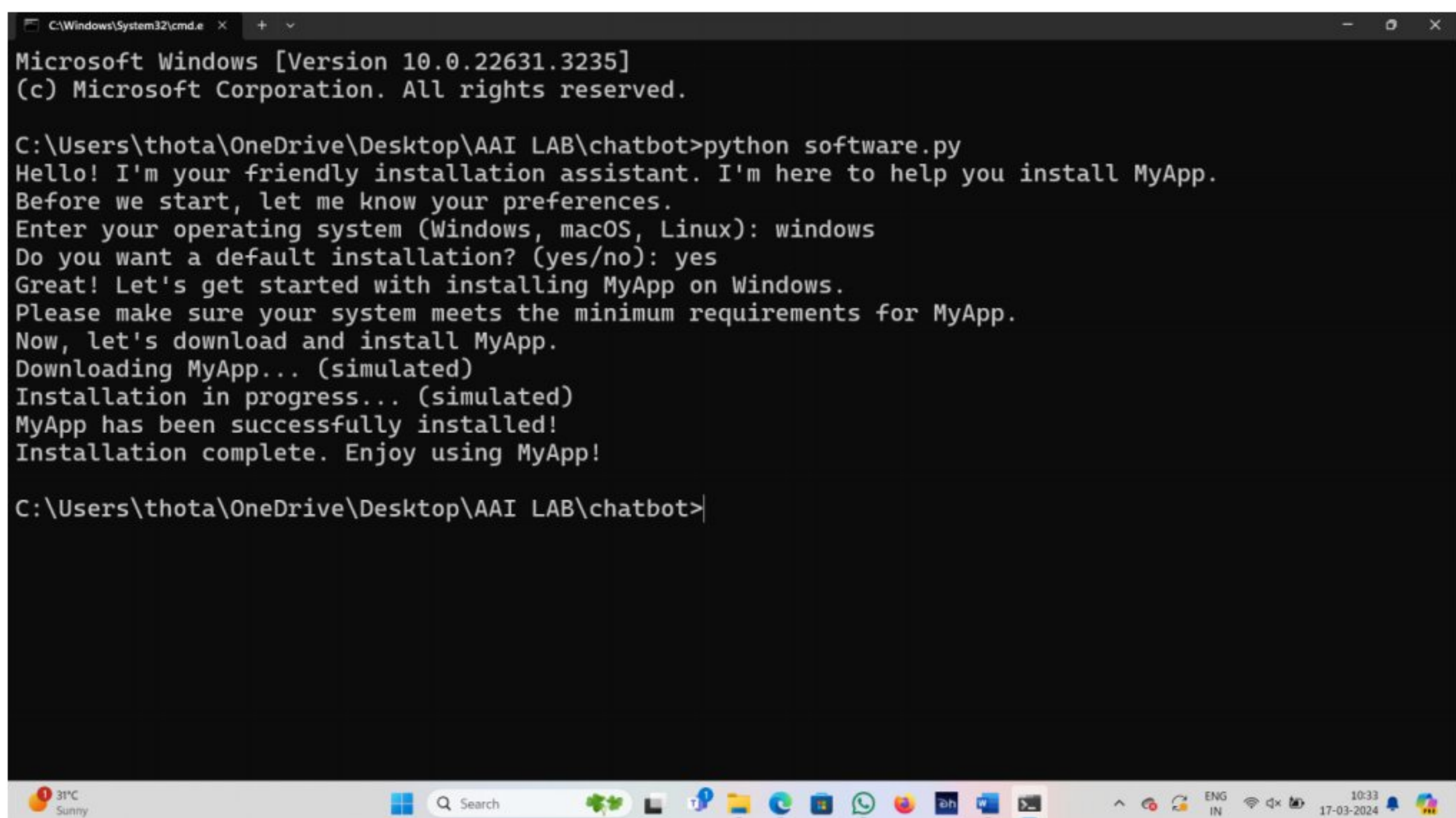
    pass

print("Installation complete. Enjoy using MyApp!")

if __name__ == "__main__":

    main()
```

OUTPUT:



```
C:\Windows\System32\cmd.e x + v
Microsoft Windows [Version 10.0.22631.3235]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\chatbot>python software.py
Hello! I'm your friendly installation assistant. I'm here to help you install MyApp.
Before we start, let me know your preferences.
Enter your operating system (Windows, macOS, Linux): windows
Do you want a default installation? (yes/no): yes
Great! Let's get started with installing MyApp on Windows.
Please make sure your system meets the minimum requirements for MyApp.
Now, let's download and install MyApp.
Downloading MyApp... (simulated)
Installation in progress... (simulated)
MyApp has been successfully installed!
Installation complete. Enjoy using MyApp!

C:\Users\thota\OneDrive\Desktop\AAI LAB\chatbot>
```

Experiment no:06	<h1>Digital Assistants</h1>
Date:	

Aim: Design a Digital assistant of your choice. Example is the digital assistant which plays a song when you say play song xxx.mp3.

Program:

```
import pyttsx3
import speech_recognition as sr
import datetime
import wikipedia
import webbrowser
import os
import random

# Initialize text-to-speech engine
engine = pyttsx3.init()

# Set voice properties (optional)
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id) # Female voice

# Function to speak
def speak(audio):
    engine.say(audio)
    engine.runAndWait()

# Function to greet user
def greet():
    hour = int(datetime.datetime.now().hour)

    if 0 <= hour < 12:
        speak("Good Morning!")

    elif 12 <= hour < 18:
        speak("Good Afternoon!")

    else:
```



```

    speak("Good Evening!")

speak("I am your digital assistant. How may I help you?")

# Function to take command

def take_command():

    recognizer = sr.Recognizer()

    with sr.Microphone() as source:

        print("Listening...")

        recognizer.pause_threshold = 1

        audio = recognizer.listen(source)

    try:

        print("Recognizing...")

        query = recognizer.recognize_google(audio, language='en-in')

        print(f"User said: {query}\n")

    except Exception as e:

        print(e)

        print("Please say that again...")

        return "None"

    return query

if __name__ == "__main__":

    greet()

    while True:

        query = take_command().lower()

        # Logic for executing tasks based on query

        if 'wikipedia' in query:

            speak('Searching Wikipedia...')

            query = query.replace("wikipedia", "")

            results = wikipedia.summary(query, sentences=2)

            speak("According to Wikipedia")

            speak(results)

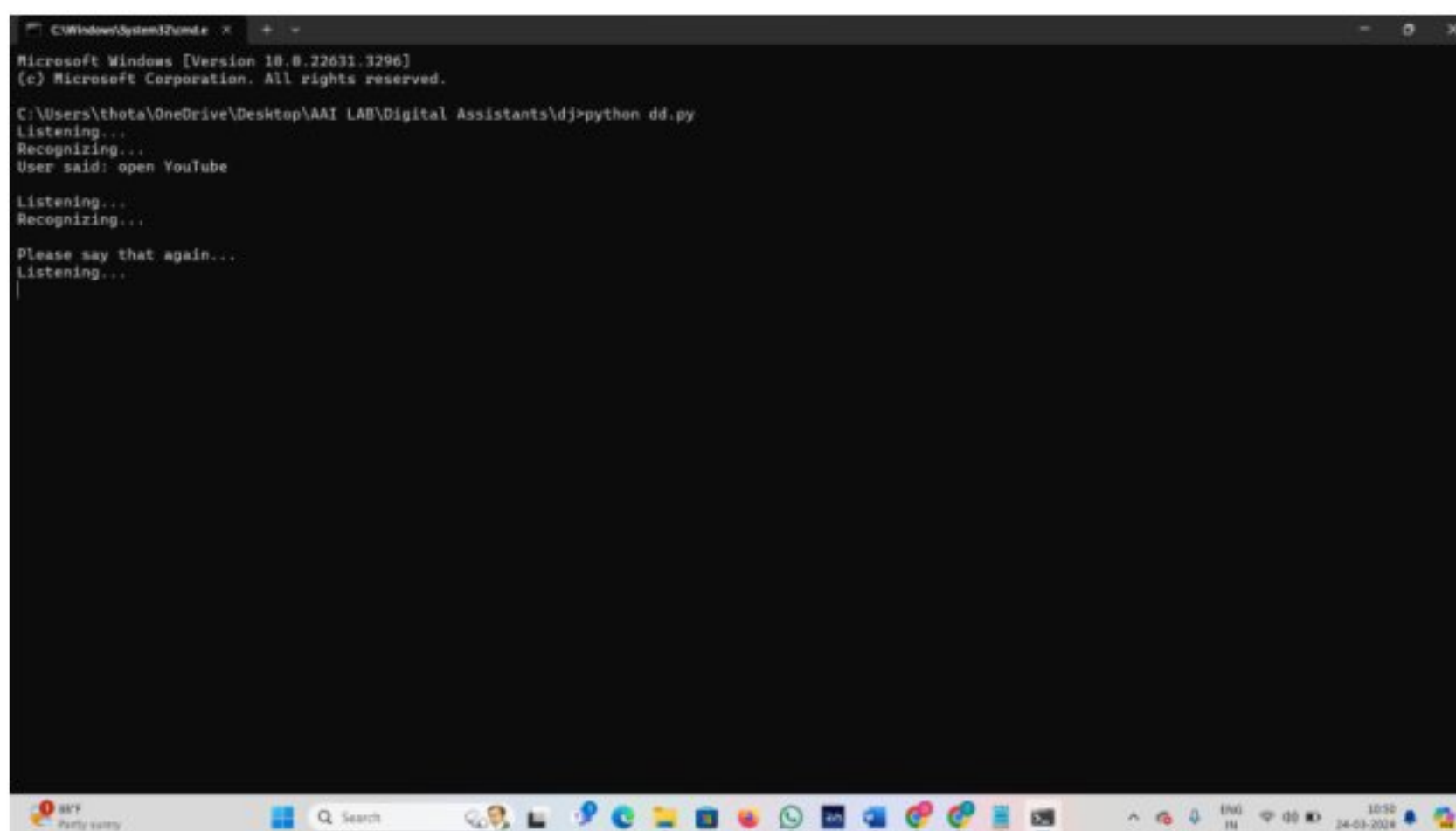
```

```

elif 'open youtube' in query:
    webbrowser.open("https://www.youtube.com")
    speak("Opening YouTube")
elif 'open google' in query:
    webbrowser.open("https://www.google.com")
    speak("Opening Google")
elif 'play music' in query:
    music_dir = 'C:\\Music' # Path to your music directory
    songs = os.listdir(music_dir)
    random_song = random.choice(songs)
    os.startfile(os.path.join(music_dir, random_song))
    speak(f"Playing {random_song}")
elif 'the time' in query:
    strTime = datetime.datetime.now().strftime("%H:%M:%S")
    speak(f"The time is {strTime}")
elif 'exit' in query:
    speak("Goodbye!")
    break
else:
    speak("I'm sorry, I don't understand that command. Please try again.")

```

OUTPUT:



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22031.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\IAI LAB\Digital Assistants\dj\python dd.py
Listening...
Recognizing...
User said: open YouTube
Listening...
Recognizing...
Please say that again...
Listening...

```

Experiment no:07	<h1>Social Media</h1>
Date:	

Aim: Design an application which detects Fake news

Program:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.metrics import accuracy_score

# Load the dataset (replace this with your own dataset)
data = pd.read_csv("news_dataset.csv")

# Separate features (X) and labels (y)
X = data['text']
y = data['label']

# Initialize a TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)

# Fit and transform the training data
X_tfidf = tfidf_vectorizer.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

# Initialize a PassiveAggressiveClassifier
classifier = PassiveAggressiveClassifier(max_iter=50)

# Train the classifier
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```

print(f"Accuracy: {accuracy:.2f}")

# Example of predicting fake news

def predict_fake_news(news_text):

    news_tfidf = tfidf_vectorizer.transform([news_text])

    prediction = classifier.predict(news_tfidf)

    if prediction[0] == 'FAKE':

        return "This news is fake."

    else:

        return "This news is real."

# Example usage

news_text = "Scientists have discovered a new planet beyond our solar system."

print(predict_fake_news(news_text))

```

OUTPUT:

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\thota\OneDrive\Desktop\AAI LAB\Social Media\New folder>python ff.py
C:\Users\thota\OneDrive\Desktop\AAI LAB\Social Media\New folder\ff.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other li
braries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

import pandas as pd
C:\Users\thota\OneDrive\Desktop\AAI LAB\Social Media\New folder\ff.py:8: DtypeWarning: Columns (4,5,6,7,8,9,10
,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,4
7,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,
84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,11
5,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,
143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,17
0,171) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('Fake.csv')
The text is with a probability of (probability:.2f)

C:\Users\thota\OneDrive\Desktop\AAI LAB\Social Media\New folder>

```

Experiment no:08	<h1>Image Processing</h1>
Date:	

Aim: Design an application which detects the emotion of the person in an image.

Program:

```
# Import the necessary libraries

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image = cv2.imread('Ganeshji.webp')

# Convert BGR image to RGB

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define the scale factor

# Increase the size by 3 times

scale_factor_1 = 3.0

# Decrease the size by 3 times

scale_factor_2 = 1/3.0

# Get the original image dimensions

height, width = image_rgb.shape[:2]

# Calculate the new image dimensions

new_height = int(height * scale_factor_1)

new_width = int(width * scale_factor_1)

# Resize the image

zoomed_image = cv2.resize(src =image_rgb,

                           dsize=(new_width, new_height),

                           interpolation=cv2.INTER_CUBIC)

# Calculate the new image dimensions

new_height1 = int(height * scale_factor_2)

new_width1 = int(width * scale_factor_2)
```

```
# Scaled image
scaled_image = cv2.resize(src= image_rgb,
                          dsize =(new_width1, new_height1),
                          interpolation=cv2.INTER_AREA)

# Create subplots
fig, axs = plt.subplots(1, 3, figsize=(10, 4))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image Shape:'+str(image_rgb.shape))

# Plot the Zoomed Image
axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image Shape:'+str(zoomed_image.shape))

# Plot the Scaled Image
axs[2].imshow(scaled_image)
axs[2].set_title('Scaled Image Shape:'+str(scaled_image.shape))

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```

INPUT:



OUTPUT:

