Q.1 Write a program to read data from a CSV file and perform basic data cleaning operations like removing duplicates or handling missing values.

import pandas as pd

def clean\_data(csv\_file):

# Read the CSV file into a DataFrame

df = pd.read\_csv(csv\_file)

# Remove duplicate rows

df.drop\_duplicates(inplace=True)

# Handling missing values

df.fillna(method='ffill', inplace=True) # Forward fill missing values

return df

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

# Path to the CSV file

csv\_file = 'data.csv'

# Clean the data

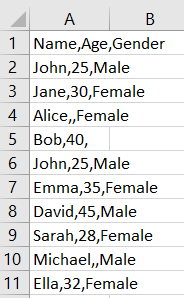
cleaned\_data = clean\_data(csv\_file)

# Display cleaned data

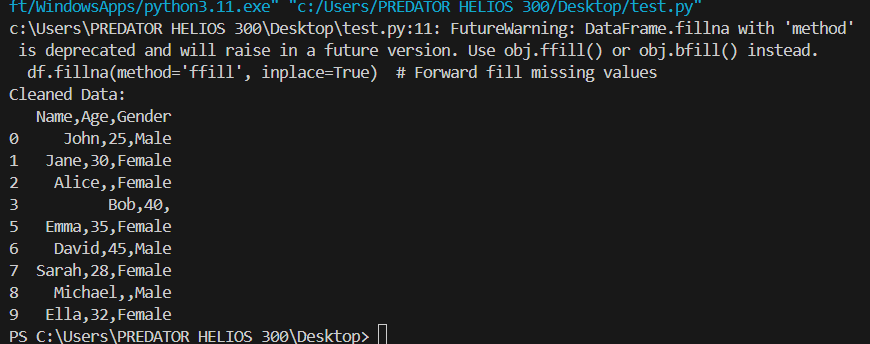
print("Cleaned Data:")

print(cleaned\_data)

Data:



Result:



Q.2 Write a program to search for a specific pattern (e.g., email addresses or phone numbers) within a text file and extract them.

import re

def extract\_emails(text):

# Define the regular expression pattern for email addresses

email\_pattern = r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

# Use the findall() function to extract email addresses from the text

emails = re.findall(email\_pattern, text)

return emails

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

# Open the text file

with open('test.txt', 'r') as file:

text = file.read()

# Extract emails from the text

extracted\_emails = extract\_emails(text)

# Display extracted emails

print("Extracted Emails:")

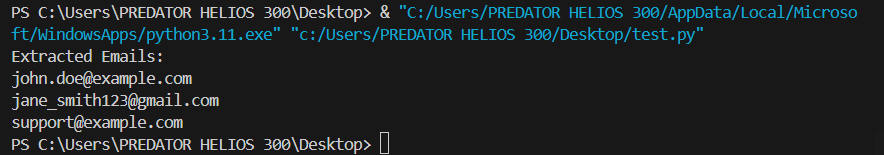
for email in extracted\_emails:

print(email)

Text file:

* This is a sample text file containing some email addresses.
* You can contact me at john.doe@example.com or jane\_smith123@gmail.com.
* Please don't hesitate to reach out to support@example.com if you have any questions.

Result:



Q.3 Write a Python program to find the shortest path in a graph using Dijkstra's algorithm.

import heapq

def dijkstra(graph, start):

# Initialize distances to all vertices as infinity

distances = {vertex: float('infinity') for vertex in graph}

distances[start] = 0 # Distance from start vertex to itself is 0

# Initialize priority queue

priority\_queue = [(0, start)] # (distance, vertex)

while priority\_queue:

# Get the vertex with the smallest distance from the priority queue

current\_distance, current\_vertex = heapq.heappop(priority\_queue)

# Check if current distance is larger than the stored distance

if current\_distance > distances[current\_vertex]:

continue

# Iterate through neighbors of the current vertex

for neighbor, weight in graph[current\_vertex].items():

distance = current\_distance + weight

# If new distance is smaller than stored distance, update it

if distance < distances[neighbor]:

distances[neighbor] = distance

heapq.heappush(priority\_queue, (distance, neighbor))

return distances

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

# Example graph (dictionary of dictionaries)

graph = {

'A': {'B': 5, 'C': 3},

'B': {'A': 5, 'C': 2, 'D': 1},

'C': {'A': 3, 'B': 2, 'D': 4},

'D': {'B': 1, 'C': 4}

}

start\_vertex = 'A'

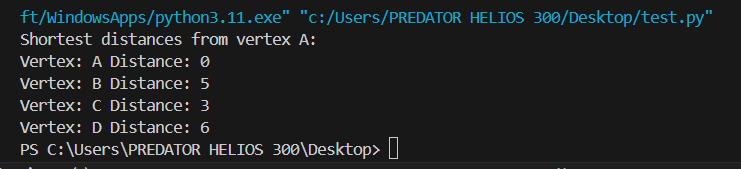
shortest\_distances = dijkstra(graph, start\_vertex)

print("Shortest distances from vertex", start\_vertex + ":")

for vertex, distance in shortest\_distances.items():

print("Vertex:", vertex, "Distance:", distance)

Result:



Q.4 Create a Python class to represent a bank account with methods for deposit, withdrawal, and checking the balance.

class BankAccount:

def \_\_init\_\_(self, account\_holder, initial\_balance=0):

self.account\_holder = account\_holder

self.balance = initial\_balance

def deposit(self, amount):

if amount > 0:

self.balance += amount

print(f"Deposited {amount} units. Current balance: {self.balance}")

else:

print("Invalid amount for deposit.")

def withdraw(self, amount):

if 0 < amount <= self.balance:

self.balance -= amount

print(f"Withdrew {amount} units. Current balance: {self.balance}")

else:

print("Insufficient funds or invalid amount for withdrawal.")

def check\_balance(self):

print(f"Current balance for account holder {self.account\_holder}: {self.balance}")

# Example usage:

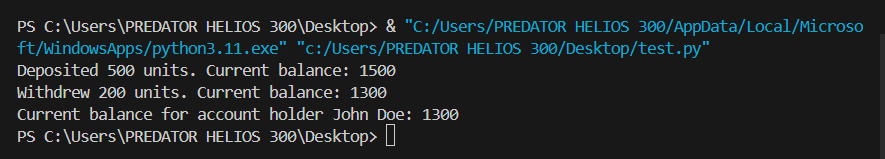
account1 = BankAccount("John Doe", 1000)

account1.deposit(500)

account1.withdraw(200)

account1.check\_balance()

Result:



Q.5 Create a GUI-based file manager that allows users to browse directories, view files, and perform basic file operations like copy, move, and delete.

import os

import shutil

import tkinter as tk

from tkinter import filedialog, messagebox

class FileManagerApp:

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

self.root = root

self.root.title("File Manager")

self.current\_path = tk.StringVar()

self.current\_path.set(os.getcwd())

self.create\_widgets()

def create\_widgets(self):

self.path\_label = tk.Label(self.root, textvariable=self.current\_path)

self.path\_label.pack()

self.file\_listbox = tk.Listbox(self.root, width=50, height=20)

self.file\_listbox.pack()

self.refresh\_button = tk.Button(self.root, text="Refresh", command=self.refresh\_files)

self.refresh\_button.pack()

self.copy\_button = tk.Button(self.root, text="Copy", command=self.copy\_file)

self.copy\_button.pack()

self.move\_button = tk.Button(self.root, text="Move", command=self.move\_file)

self.move\_button.pack()

self.delete\_button = tk.Button(self.root, text="Delete", command=self.delete\_file)

self.delete\_button.pack()

self.browse\_button = tk.Button(self.root, text="Browse", command=self.browse\_directory)

self.browse\_button.pack()

self.refresh\_files()

def browse\_directory(self):

directory = filedialog.askdirectory()

if directory:

self.current\_path.set(directory)

self.refresh\_files()

def refresh\_files(self):

self.file\_listbox.delete(0, tk.END)

path = self.current\_path.get()

for item in os.listdir(path):

self.file\_listbox.insert(tk.END, item)

def get\_selected\_file(self):

selection = self.file\_listbox.curselection()

if selection:

return self.file\_listbox.get(selection[0])

else:

return None

def copy\_file(self):

filename = self.get\_selected\_file()

if filename:

src = os.path.join(self.current\_path.get(), filename)

dst = filedialog.asksaveasfilename(initialdir=os.getcwd(), initialfile=filename)

if dst:

shutil.copy2(src, dst)

messagebox.showinfo("Success", f"File '{filename}' copied successfully.")

def move\_file(self):

filename = self.get\_selected\_file()

if filename:

src = os.path.join(self.current\_path.get(), filename)

dst = filedialog.askdirectory()

if dst:

shutil.move(src, dst)

self.refresh\_files()

messagebox.showinfo("Success", f"File '{filename}' moved successfully.")

def delete\_file(self):

filename = self.get\_selected\_file()

if filename:

if messagebox.askyesno("Delete", f"Are you sure you want to delete '{filename}'?"):

path = os.path.join(self.current\_path.get(), filename)

os.remove(path)

self.refresh\_files()

messagebox.showinfo("Success", f"File '{filename}' deleted successfully.")

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

root = tk.Tk()

app = FileManagerApp(root)

root.mainloop()

Result:

