1. class MaxHeap:

    def \_\_init\_\_(self):

        self.heap = []

    def insert(self, value):

        self.heap.append(value)

        self.\_heapify\_up(len(self.heap) - 1)

    def delete(self):

        if len(self.heap) == 0:

            raise IndexError("Oops! The heap is empty. Nothing to delete.")

        max\_value = self.heap[0]

        last\_value = self.heap.pop()

        if len(self.heap) > 0:

            self.heap[0] = last\_value

            self.\_heapify\_down(0)

        return max\_value

    def get\_max(self):

        if len(self.heap) == 0:

            raise IndexError("Oops! The heap is empty. No maximum element available.")

        return self.heap[0]

    def \_heapify\_up(self, index):

        while index > 0:

            parent\_index = (index - 1) // 2

            if self.heap[parent\_index] < self.heap[index]:

                self.heap[parent\_index], self.heap[index] = self.heap[index], self.heap[parent\_index]

                index = parent\_index

            else:

                break

    def \_heapify\_down(self, index):

        while True:

            left\_child\_index = 2 \* index + 1

            right\_child\_index = 2 \* index + 2

            largest\_index = index

            if left\_child\_index < len(self.heap) and self.heap[left\_child\_index] > self.heap[largest\_index]:

                largest\_index = left\_child\_index

            if right\_child\_index < len(self.heap) and self.heap[right\_child\_index] > self.heap[largest\_index]:

                largest\_index = right\_child\_index

            if largest\_index != index:

                self.heap[largest\_index], self.heap[index] = self.heap[index], self.heap[largest\_index]

                index = largest\_index

            else:

                break

max\_heap = MaxHeap()

max\_heap.insert(5)

max\_heap.insert(10)

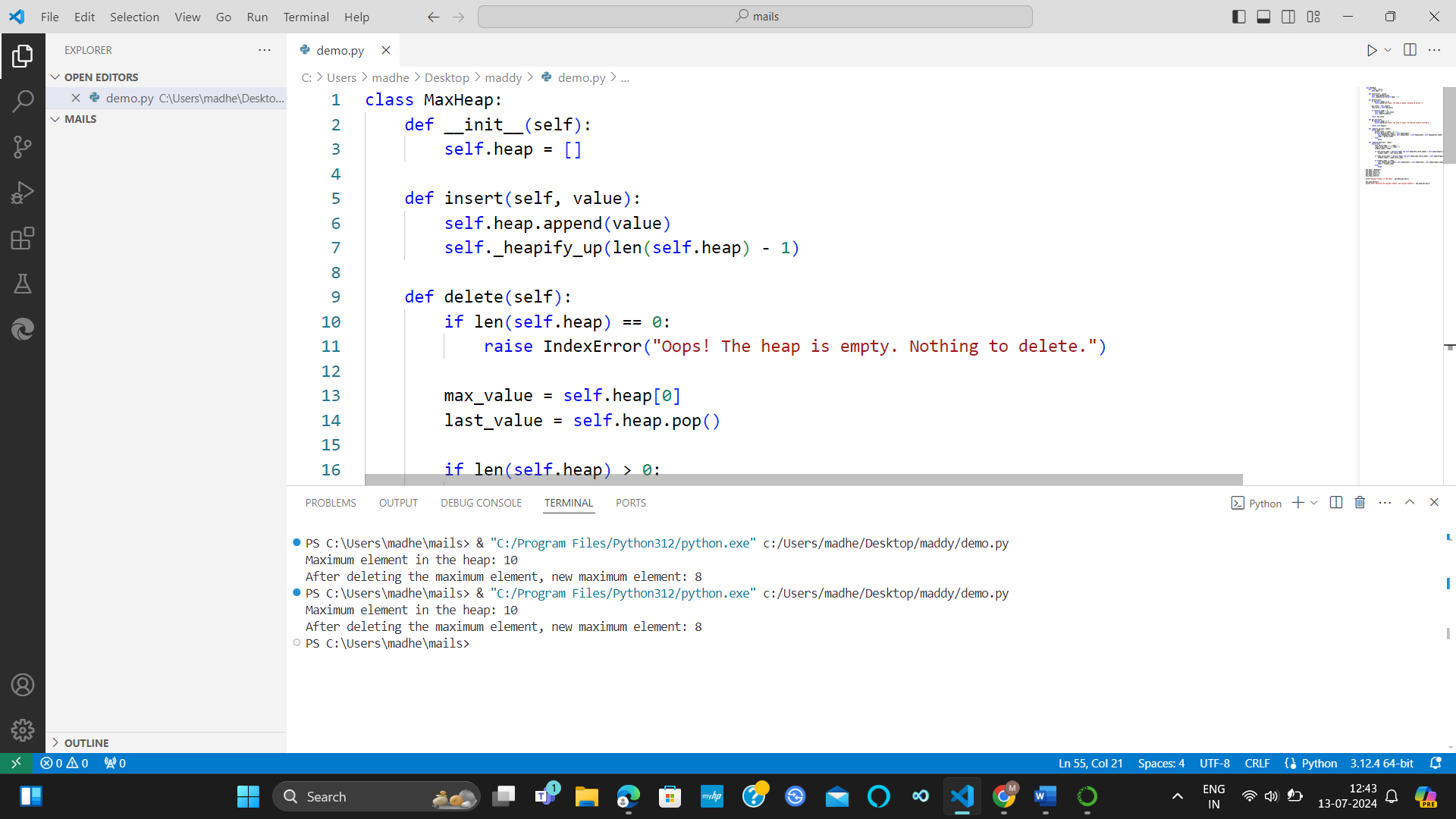
max\_heap.insert(3)

max\_heap.insert(8)

print("Maximum element in the heap:", max\_heap.get\_max())

max\_heap.delete()

print("After deleting the maximum element, new maximum element:", max\_heap.get\_max())



2. import requests

from time import sleep

def download\_urls(urls, max\_retries=4):

    results = {}

    for url in urls:

        attempt = 0

        while attempt < max\_retries:

            try:

                res = requests.get(url)

                res.raise\_for\_status()

                results[url] = res.content.decode('utf-8')

                break

            except requests.exceptions.RequestException as e:

                attempt += 1

                if attempt == max\_retries:

                    results[url] = f"Error downloading {url}: {str(e)}"

                else:

                    sleep(2)

    return results

urls = ['https://example.com'

]

results = download\_urls(urls)

for url, content in results.items():

    print(f"URL: {url}")

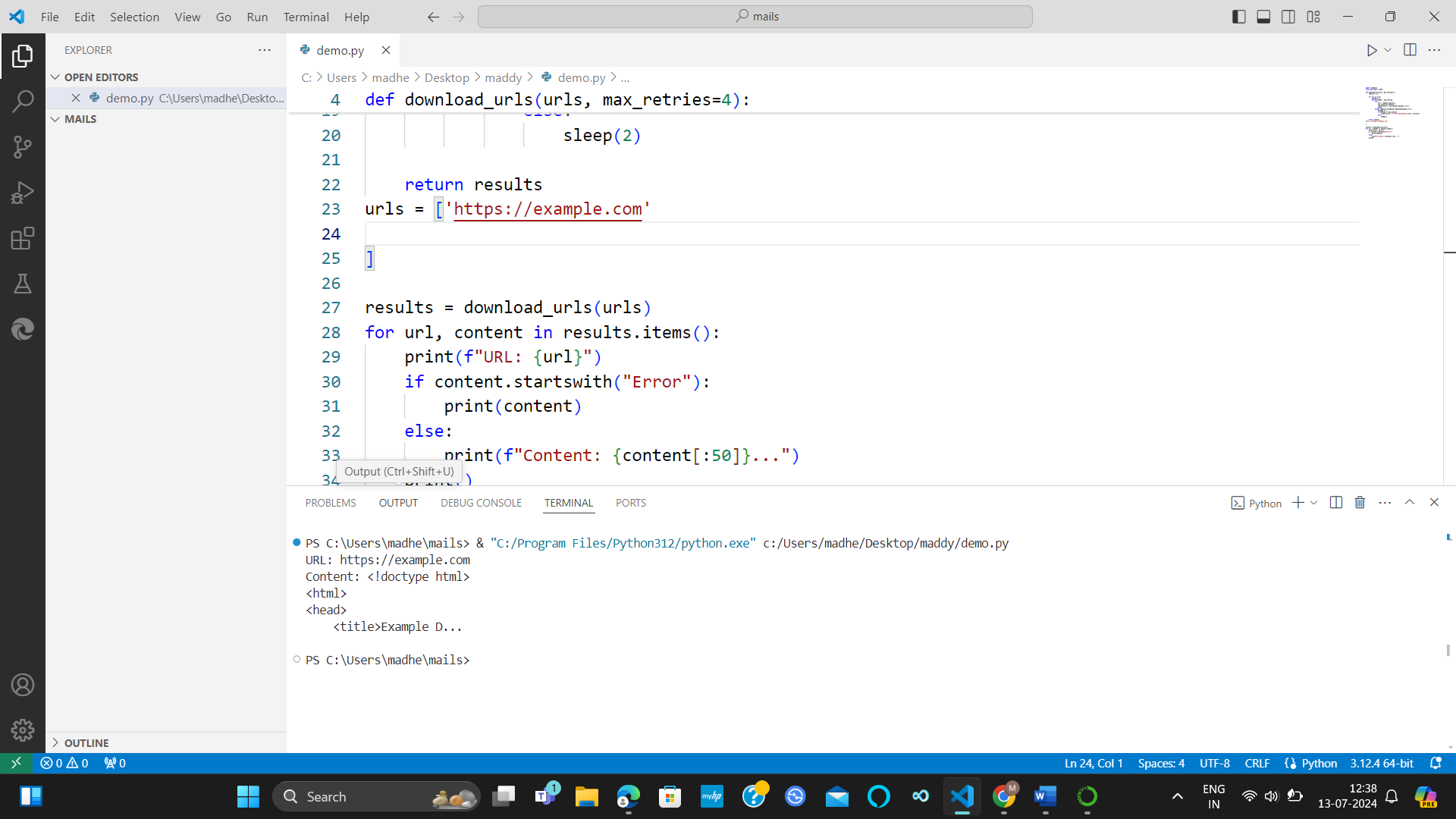
    if content.startswith("Error"):

        print(content)

    else:

        print(f"Content: {content[:50]}...")

    print()



3. import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

df=pd.read\_csv("carprices.csv")

plt.xlabel("Mileage")

plt.ylabel("Sell Price($)")

plt.scatter(df[['Mileage']],df[['Sell Price($)']])

plt.show()

x=df[['Age(yrs)','Mileage']]

y=df[['Sell Price($)']]

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=10)

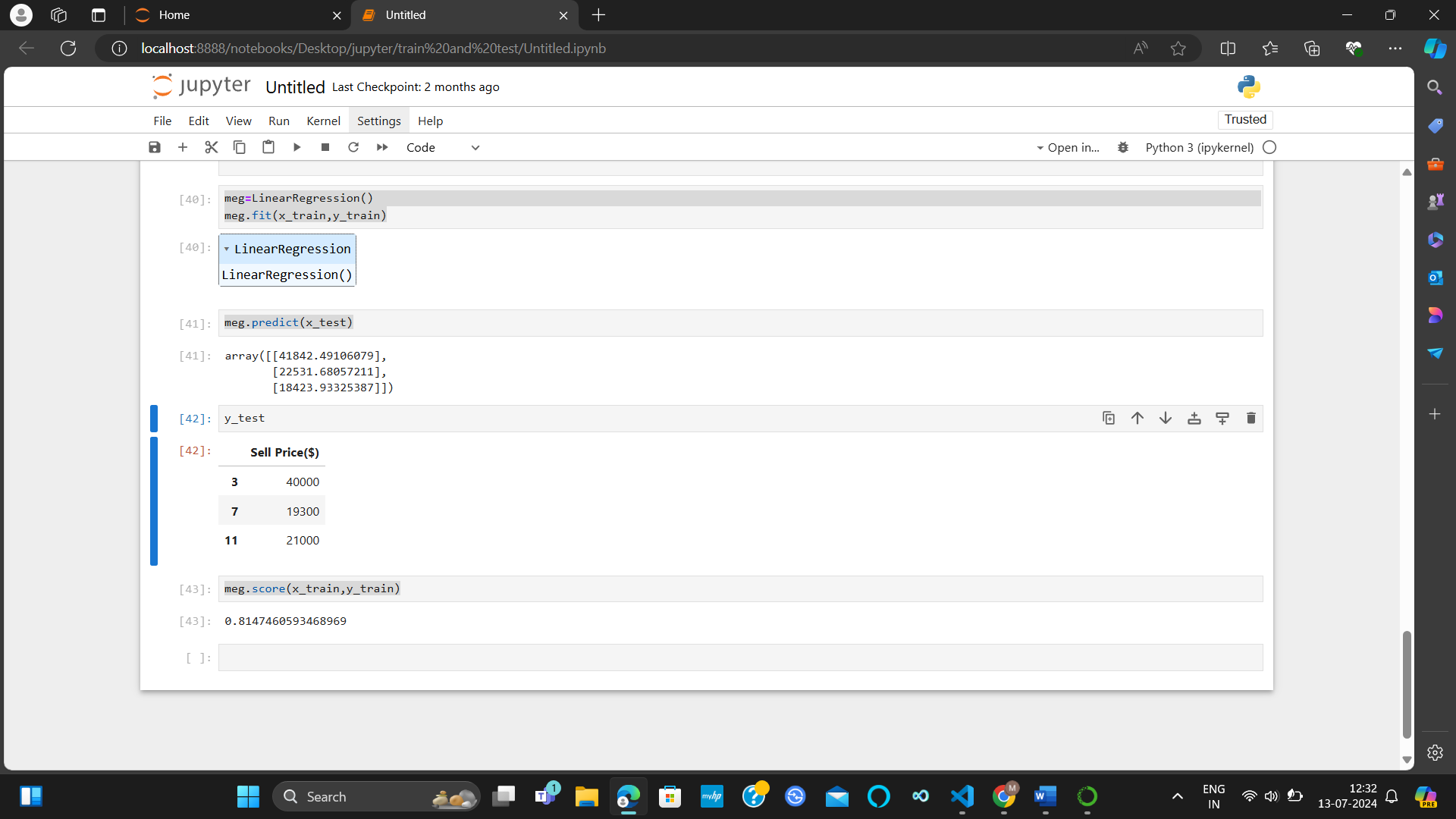
len(x\_train)

meg=LinearRegression()

meg.fit(x\_train,y\_train)

meg.predict(x\_test)

meg.score(x\_train,y\_train)



4. import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import  linear\_model

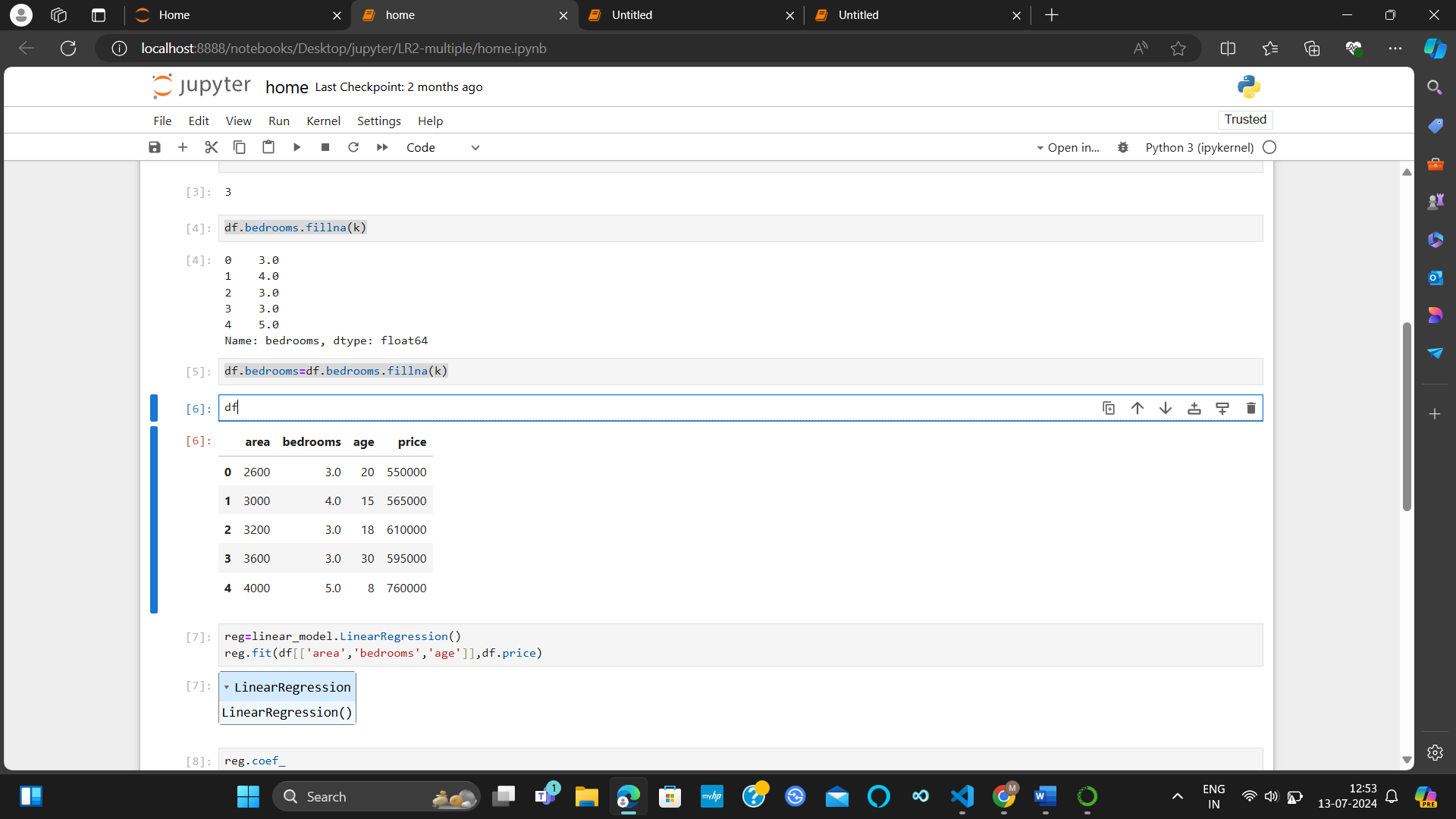
df=pd.read\_csv('homeprices.csv')

import math

k=math.floor(df.bedrooms.median())

k

df.bedrooms.fillna(k)



5. def fib(n):

    if n <= 0:

        return "Pls enter the Positive Value."

    elif n == 1:

        return 0

    elif n == 2:

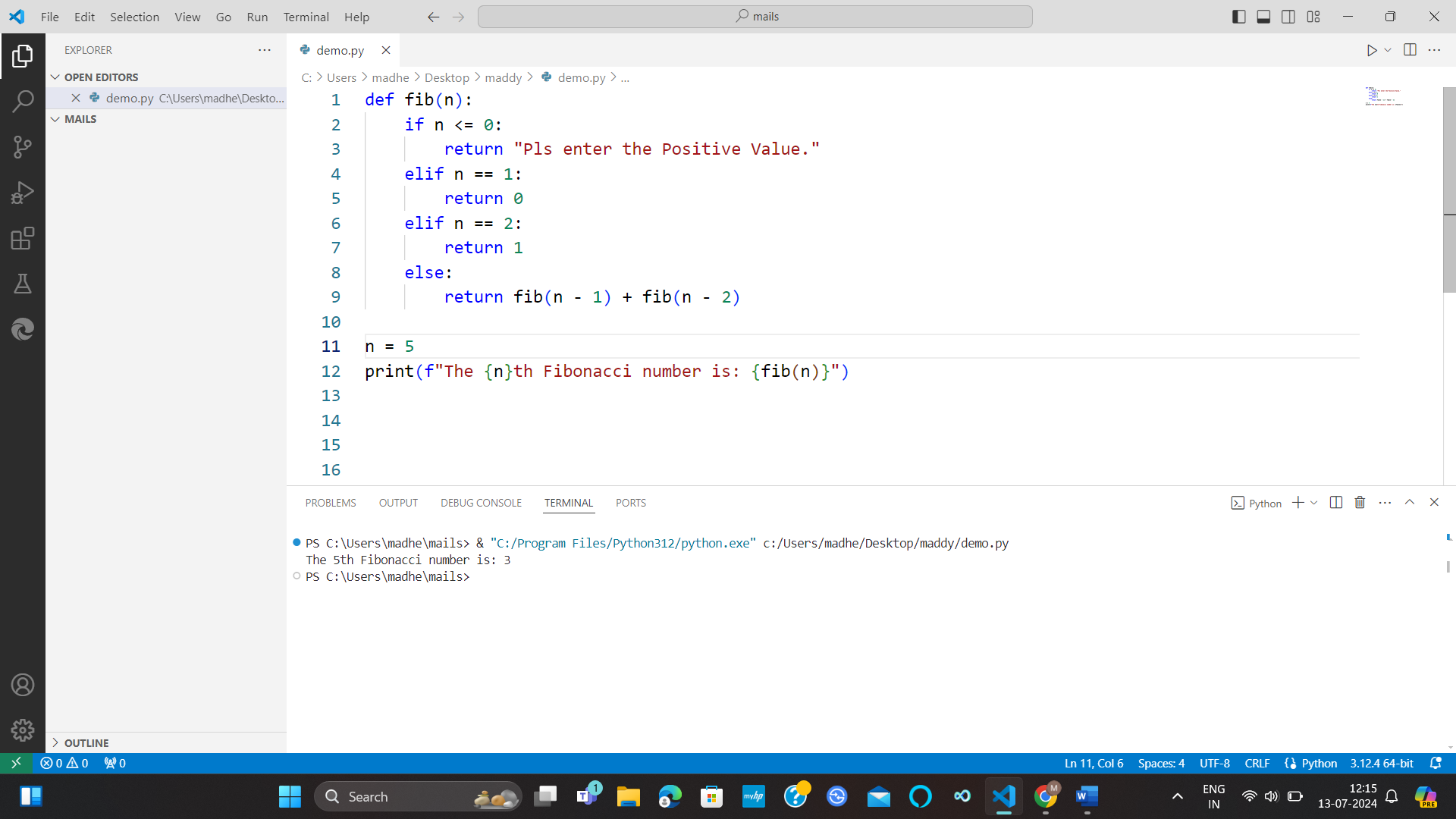
        return 1

    else:

        return fib(n - 1) + fib(n - 2)

n = 5

print(f"The {n}th Fibonacci number is: {fib(n)}")



6. def divide(num1, num2):

    try:

        result = num1 / num2

    except ZeroDivisionError:

        return "Error: Division by zero is not allowed."

    except TypeError:

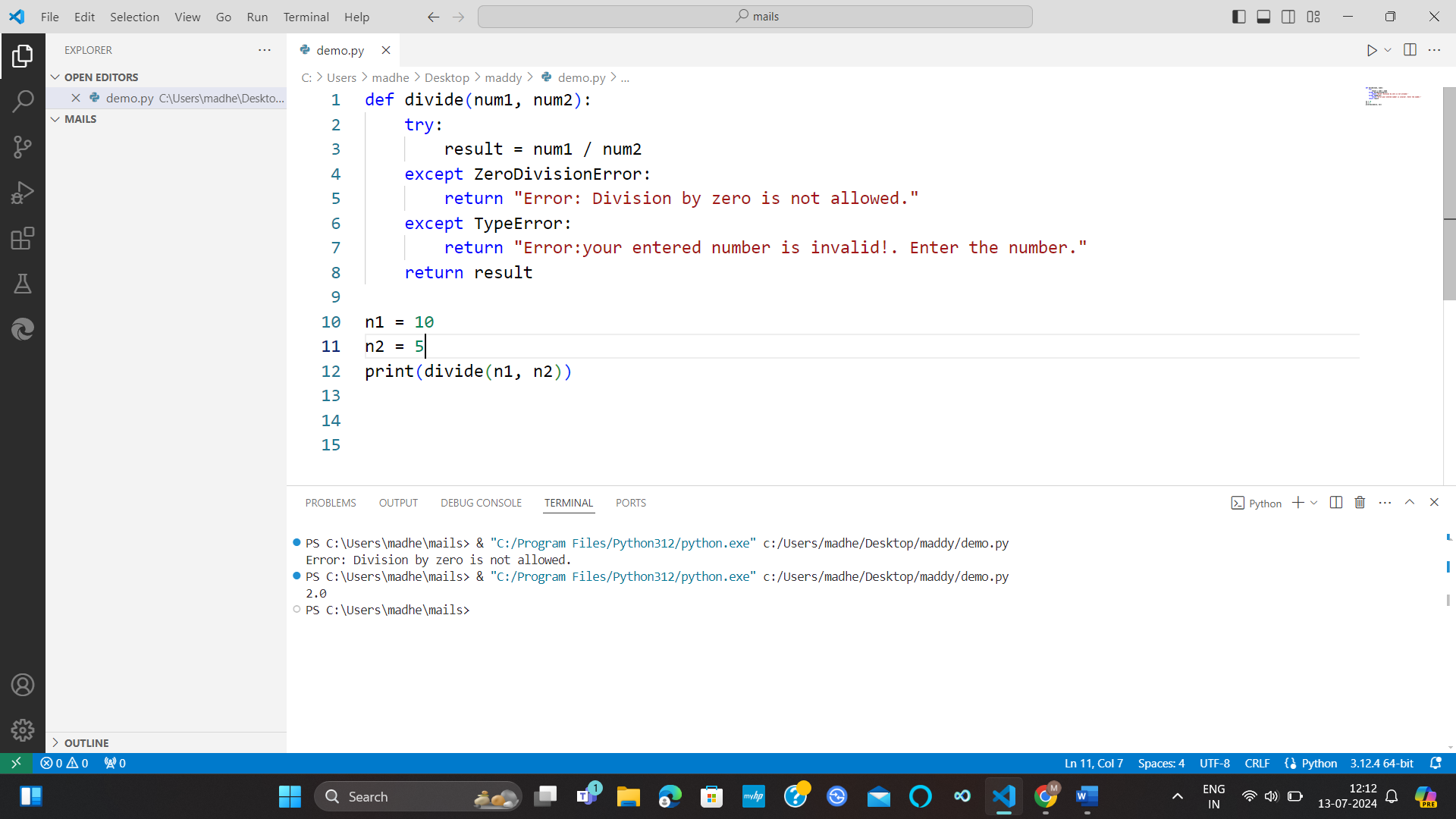
        return "Error:your entered number is invalid!. Enter the number."

    return result

n1 = 10

n2 = 5

print(divide(n1, n2))



7. import time

import logging

logging.basicConfig(level=logging.INFO)

def execution\_time\_logger(func):

    def wrapper(\*args, \*\*kwargs):

        start\_time = time.time()

        result = func(\*args, \*\*kwargs)

        end\_time = time.time()

        execution\_time = end\_time - start\_time

        logging.info(f"Executed {func.\_\_name\_\_} in {execution\_time:.4f} seconds")

        return result

    return wrapper

@execution\_time\_logger

def expensive\_computation(n):

    total = 0

    for i in range(n):

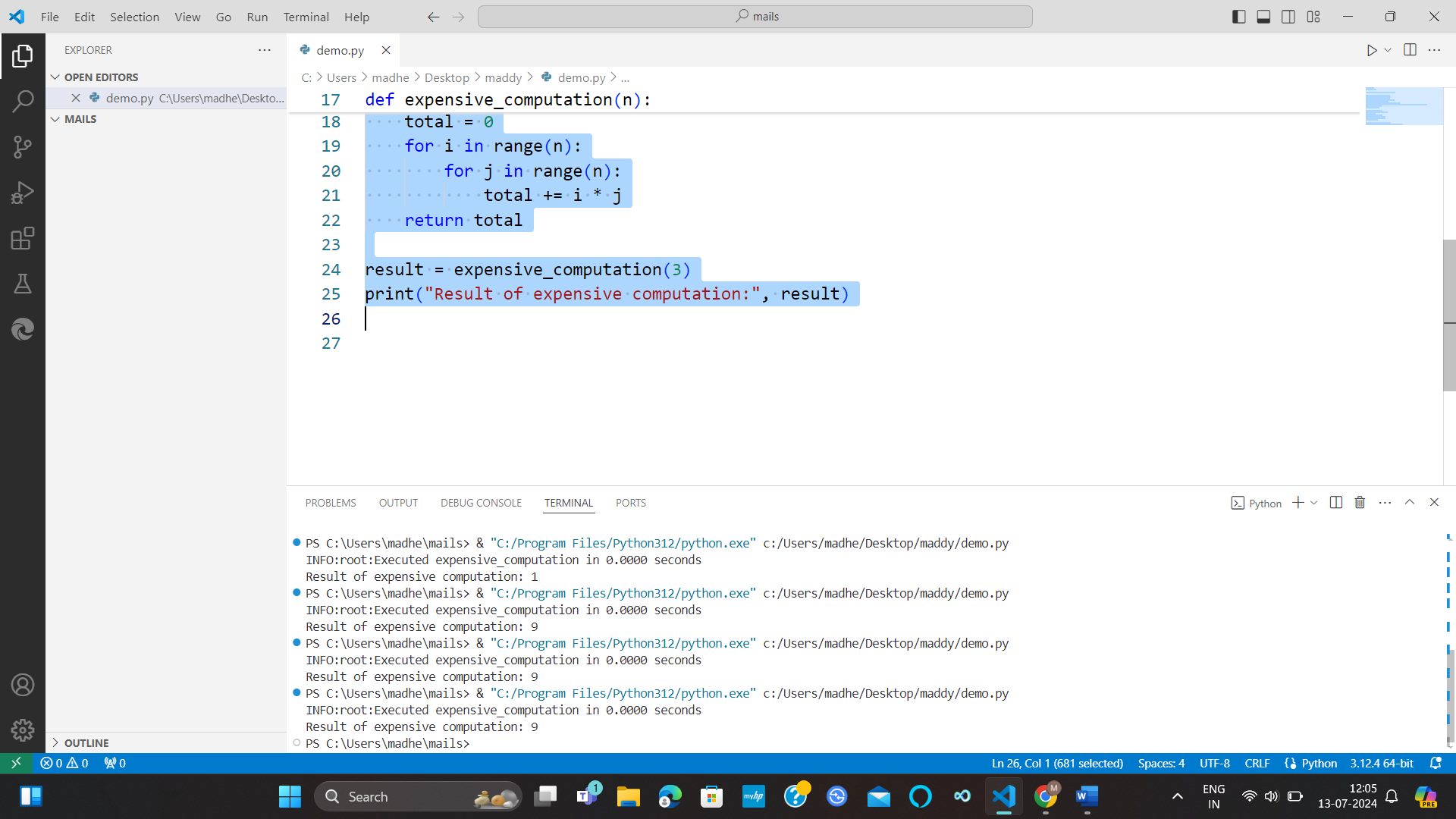
        for j in range(n):

            total += i \* j

    return total

result = expensive\_computation(3)

print("Result of expensive computation:", result)



8.def calculate(num1, num2, operator):

    if operator == '+':

        return num1 + num2

    elif operator == '-':

        return num1 - num2

    elif operator == '\*':

        return num1 \* num2

    elif operator == '/':

        if num2 == 0:

            return "Error: Division by zero"

        return num1 / num2

    else:

        return "Error: Invalid operator,pls choose the valid operator"

try:

    n1 = float(input("Enter the first number: "))

    n2 = float(input("Enter the second number: "))

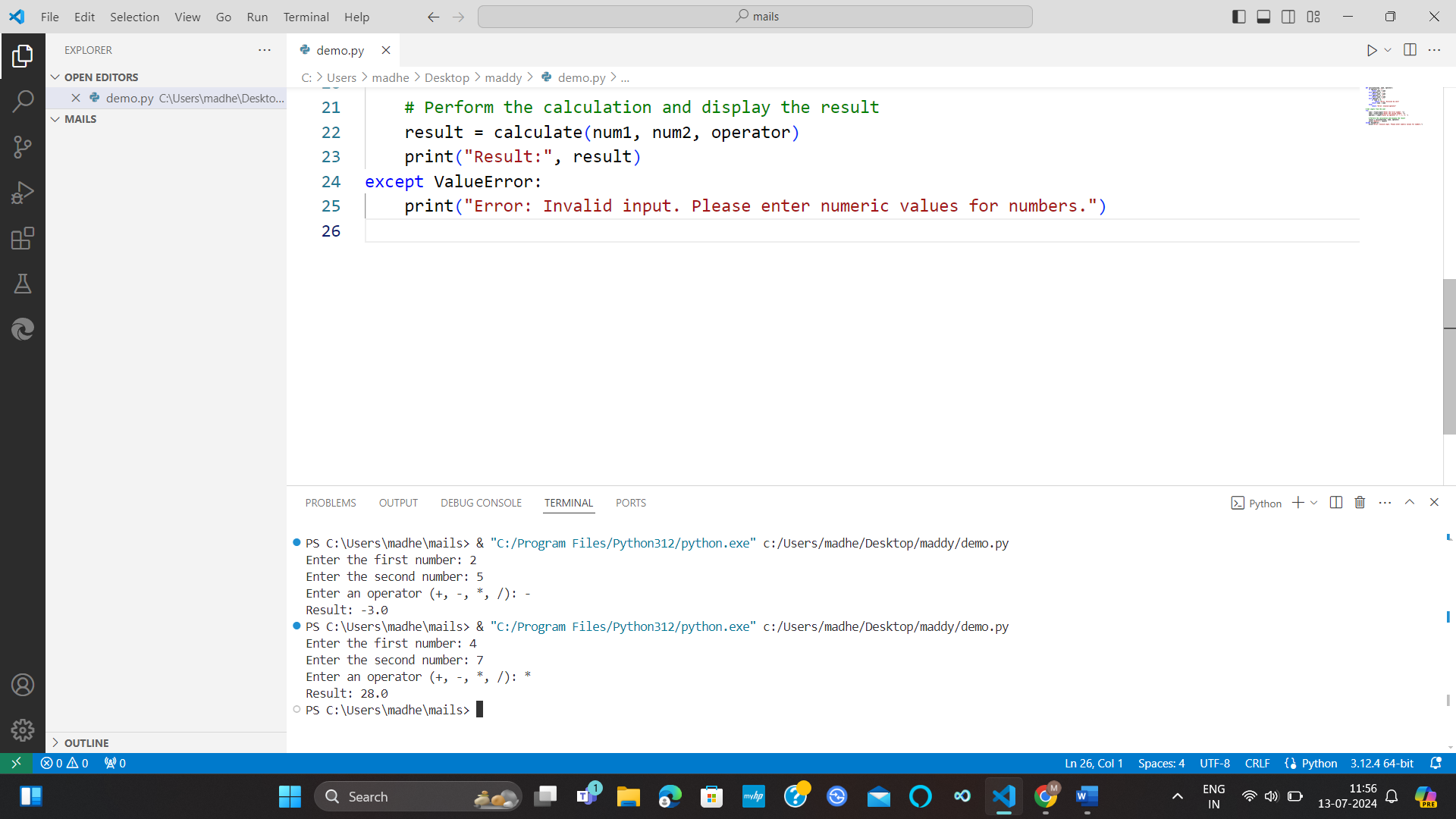
    operator = input("Enter any one of the operator (+, -, \*, /): ")

    result = calculate(n1, n2, operator)

    print(result)

except ValueError:

    print("Error: Invalid input. Please enter numbers."



9. import random

import string

def generate\_random\_password(length):

    if length < 5:

        raise ValueError("Password length is less than 5")

    uppercase = string.ascii\_uppercase

    lowercase = string.ascii\_lowercase

    digits = string.digits

    special\_char = string.punctuation

    password = [

        random.choice(uppercase),

        random.choice(lowercase),

        random.choice(digits),

        random.choice(special\_char)

    ]

    all\_characters = uppercase + lowercase + digits + special\_char

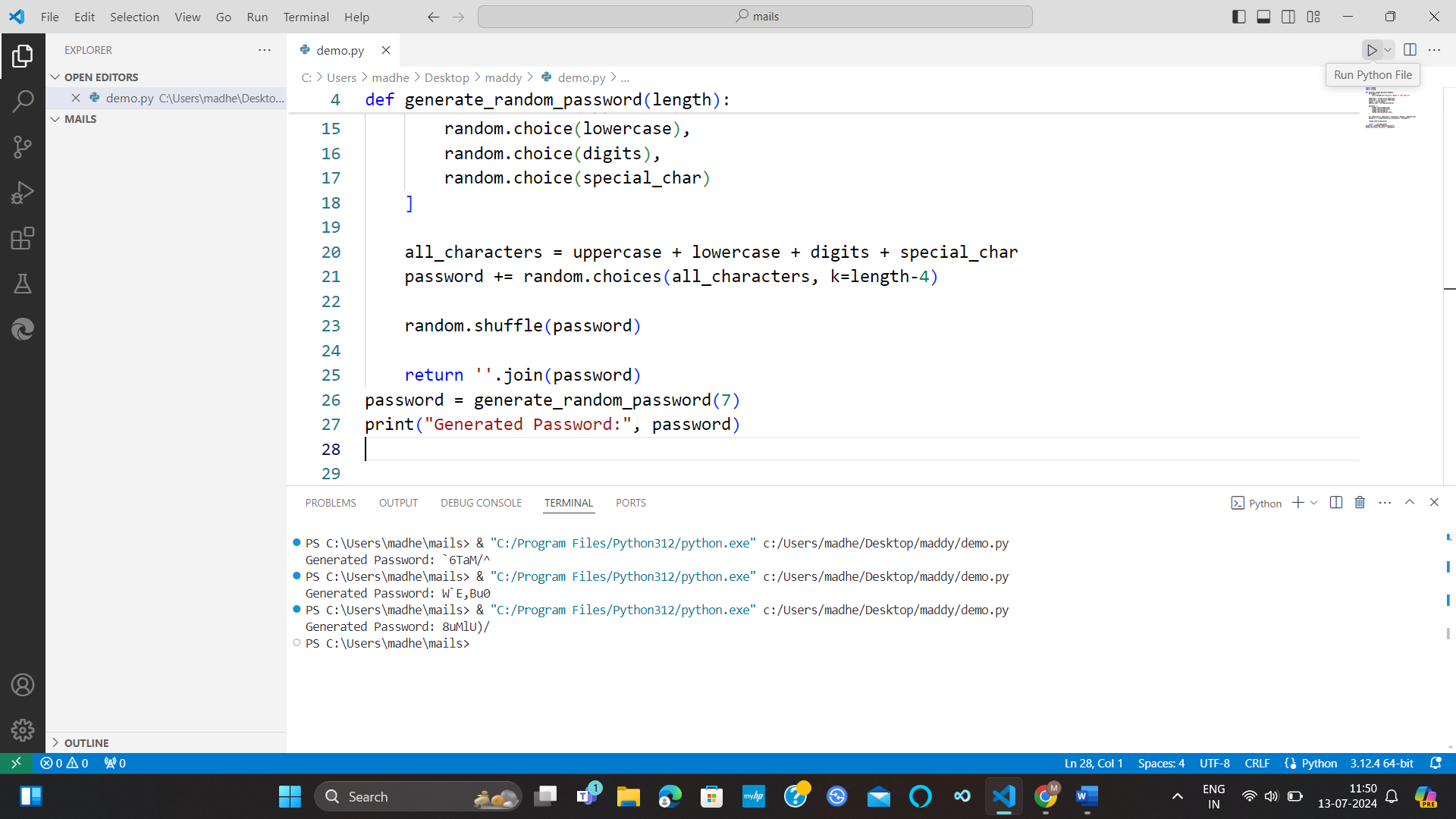
    password += random.choices(all\_characters, k=length-4)

    random.shuffle(password)

    return ''.join(password)

password = generate\_random\_password(7)

print("Generated Password:", password)



10. def transpose(matrix):

    rows = len(matrix)

    cols = len(matrix[0])

    trans = []

    for col in range(cols):

        new\_row = []

        for row in range(rows):

            new\_row.append(matrix[row][col])

        trans.append(new\_row)

    return trans

matrix = [

    [1, 2],

    [4, 5]

]

transposed\_matrix = transpose(matrix)

print("Transposed Matrix:")

for row in transposed\_matrix:

    print(row)

