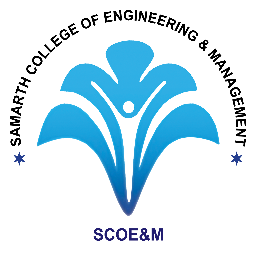
**Certificate**

This is certify to the work embodies in the “LP-3 laboratory” practical.This are bonafide students of this institute and the work has

been carried out by them under the guidance of “prof. Pansare P.P. ” and it is approved for the partial full- fillment of the requirement of savitribai phule pune university for the degree of bachelor of engineering third year of computer engineering.

****

**Name: Padekar Sayoni Prabhakar**

**Roll No: 41**

**Prof.Pansare P.P. Prof.Shegar S.R.**

**(Dept.Of Comp.Engg) (HOD Of Comp. Engg)**

**Dr. Narawade N.S**

**(Principle of SGOI COE)**

**DAA**

**ASSIGNMENT NO.1**

**Title: Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity**

**Code:**

public class Fibonacci {

    public static int fibonacciRecursive(int n)

        if (n <= 1) {

            return n;

        }

        return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);

    }

    public static int fibonacciIterative(int n) {

        if (n <= 1) {

            return n;

        }

        int a = 0, b = 1, c = 0;

        for (int i = 2; i <= n; i++) {

            c = a + b:

            a = b;

            b = c;

        }

        return c;

    }

    public static void main(String[] args) {

        int n = 10;

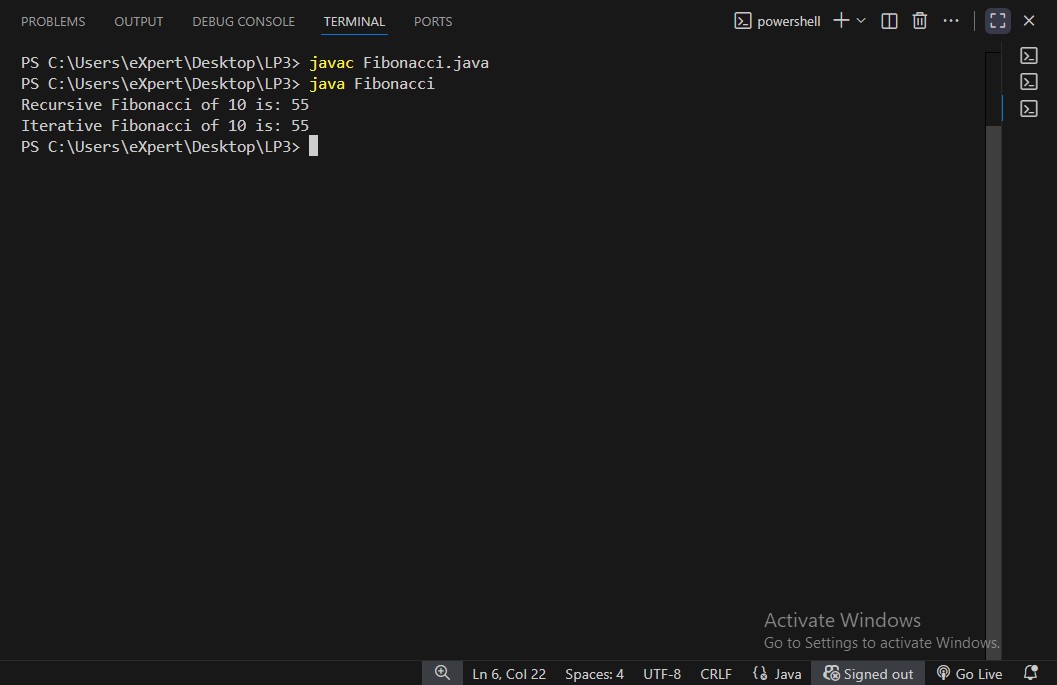
        System.out.println("Recursive Fibonacci of " + n + " is: " + fibonacciRecursive(n));

        System.out.println("Iterative Fibonacci of " + n + " is: " + fibonacciIterative(n));

    }

}

**Output:**



**ASSIGNMENT NO.2**

**Title:** **. Write a program to implement Huffman Encoding using a greedy strategy.**

**Code:**

import java.util.\*;

class HuffmanNode {

    char character;

    int frequency;

    HuffmanNode left;

    HuffmanNode right;

    HuffmanNode(char character, int frequency) {

        this.character = character;

        this.frequency = frequency;

        this.left = null;

        this.right = null;

    }

    HuffmanNode(int frequency, HuffmanNode left, HuffmanNode right) {

        this.character = '\0';

        this.frequency = frequency;

        this.left = left;

        this.right = right;

    }

}

class FrequencyComparator implements Comparator<HuffmanNode> {

    public int compare(HuffmanNode a, HuffmanNode b) {

        return a.frequency - b.frequency;

    }

}

public class HuffmanEncoding {

    public static HuffmanNode buildHuffmanTree(Map<Character, Integer> frequencyMap) {

        PriorityQueue<HuffmanNode> pq = new PriorityQueue<>(new FrequencyComparator());

        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

            pq.add(new HuffmanNode(entry.getKey(), entry.getValue()));

        }

        while (pq.size() > 1) {

            HuffmanNode left = pq.poll();

            HuffmanNode right = pq.poll();

            int sumFreq = left.frequency + right.frequency;

            HuffmanNode parent = new HuffmanNode(sumFreq, left, right);

            pq.add(parent);

        }

        return pq.poll();

    }

    public static void generateCodes(HuffmanNode root, String code, Map<Character, String> huffmanCodeMap) {

        if (root == null) return;

        if (root.left == null && root.right == null && root.character != '\0') {

            huffmanCodeMap.put(root.character, code);

        }

        generateCodes(root.left, code + "0", huffmanCodeMap);

        generateCodes(root.right, code + "1", huffmanCodeMap);

    }

    public static void printHuffmanCodes(Map<Character, String> huffmanCodeMap) {

        System.out.println("Character\tHuffman Code");

        for (Map.Entry<Character, String> entry : huffmanCodeMap.entrySet()) {

            System.out.println(entry.getKey() + "\t\t" + entry.getValue());

        }

    }

    public static void main(String[] args) {

        String text = "huffman encoding in java";

        Map<Character, Integer> frequencyMap = new HashMap<>();

        for (char c : text.toCharArray()) {

            if (c == ' ') continue;

            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

        }

        HuffmanNode root = buildHuffmanTree(frequencyMap);

        Map<Character, String> huffmanCodeMap = new HashMap<>();

        generateCodes(root, "", huffmanCodeMap);

        printHuffmanCodes(huffmanCodeMap);

        StringBuilder encoded = new StringBuilder();

        for (char c : text.toCharArray()) {

            if (c == ' ') continue;

            encoded.append(huffmanCodeMap.get(c));

        }

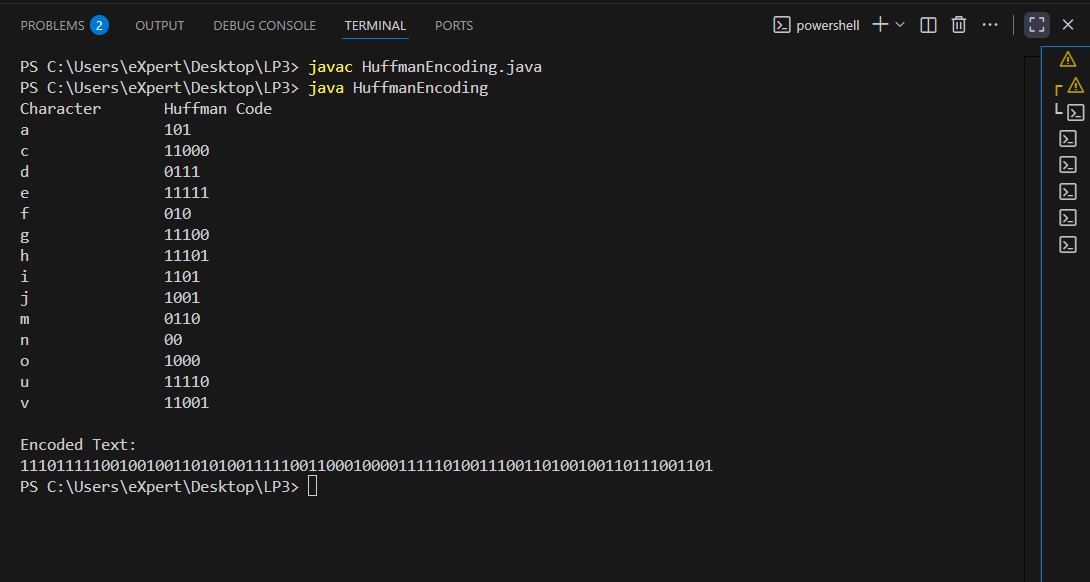
        System.out.println("\nEncoded Text:");

        System.out.println(encoded.toString());

    }

}

**Output:**

****

**ASSIGNMENT NO.3**

**Title:** **Write a program to solve a fractional Knapsack problem using a greedy method.**

**Code:**

import java.util.Arrays;

import java.util.Comparator;

class Item {

    int value, weight;

    public Item(int value, int weight) {

        this.value = value;

        this.weight = weight;

    }

    public double getRatio() {

        return (double) value / weight;

}

public class FractionalKnapsack {

    public static double fractionalKnapsack(int capacity, Item[] items) {

        Arrays.sort(items, Comparator.comparingDouble(Item::getRatio).reversed());

        double totalValue = 0.0;

        int remainingCapacity = capacity;

        for (Item item : items) {

            if (item.weight <= remainingCapacity) {

                remainingCapacity -= item.weight;

                totalValue += item.value;

            } else {

                double fraction = (double) remainingCapacity / item.weight;

                totalValue += item.value \* fraction;

                break;

            }

        }

        return totalValue;

    }

    public static void main(String[] args) {

        Item[] items = {

            new Item(60, 10),

            new Item(100, 20),

            new Item(120, 30)

        };

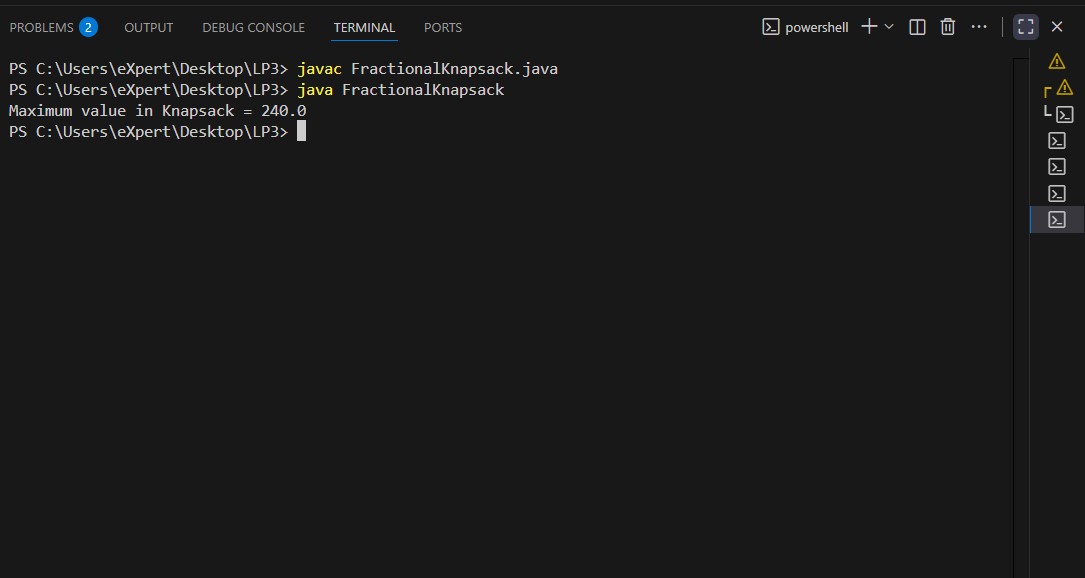
        int capacity = 50;

        double maxValue = fractionalKnapsack(capacity, items);

        System.out.println("Maximum value in Knapsack = " + maxValue);

    }

**}**

****

**ASSIGNMENT NO.4**

**Title:** **Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen’s matrix.**

**Code:**

public class NQueensWithFirstPlaced {

    static int N;

    static void printBoard(int[][] board) {

        for (int[] row : board) {

            for (int cell : row) {

                System.out.print(cell + " ");

            }

            System.out.println();

        }

        System.out.println();

    }

    static boolean isSafe(int[][] board, int row, int col) {

        int i, j;

        for (i = 0; i < row; i++)

            if (board[i][col] == 1)

                return false;

        for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

            if (board[i][j] == 1)

                return false;

        for (i = row, j = col; i >= 0 && j < N; i--, j++)

            if (board[i][j] == 1)

                return false;

        return true;

    }

    static boolean solveNQueens(int[][] board, int row) {

        if (row == N)

            return true;

        for (int col = 0; col < N; col++) {

            if (board[row][col] == 1) {

                if (solveNQueens(board, row + 1))

                    return true;

            } else if (isSafe(board, row, col)) {

                board[row][col] = 1;

                if (solveNQueens(board, row + 1))

                    return true;

                board[row][col] = 0;

            }

        }

        return false;

    }

    public static void main(String[] args) {

        N = 8;

        int[][] board = new int[N][N];

        board[0][0] = 1;

        if (solveNQueens(board, 1)) {

            System.out.println("Final N-Queens solution:");

            printBoard(board);

        } else {

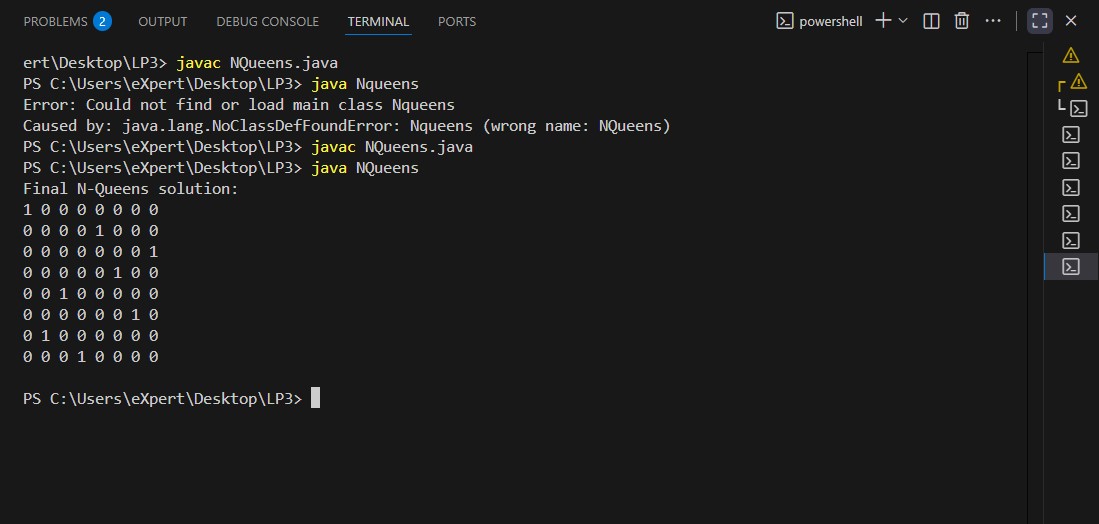
            System.out.println("No solution exists for N = " + N);

        }

    }

}

**Output:**

****

**ASSIGNMENT NO.5**

**Title: Write a program for analysis of quick sort by using deterministic and randomized variant.**

**Code:**

import java.util.\*;

class RandomizedQsort

{

    static void chooseRandomPivot(int arr[], int low, int high)

    {

        Random rand = new Random();

        int pivot = rand.nextInt(high - low + 1) + low;

        int temp = arr[pivot];

        arr[pivot] = arr[high];

        arr[high] = temp;

    }

    static int partition(int arr[], int low, int high)

    {

        chooseRandomPivot(arr, low, high);

        int pivot = arr[high];

        int i = low - 1;

        for (int j = low; j < high; j++)

        {

            if (arr[j] < pivot)

            {

                i++;

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

        int temp = arr[i + 1];

        arr[i + 1] = arr[high];

        arr[high] = temp;

        return i + 1;

    }

    static void sort(int arr[], int low, int high)

    {

        if (low < high)

        {

            int pi = partition(arr, low, high);

            sort(arr, low, pi - 1);

            sort(arr, pi + 1, high);

        }

    }

    static void printArray(int arr[])

    {

        for (int num : arr)

            System.out.print(num + " ");

        System.out.println();

    }

    public static void main(String args[])

    {

        int arr[] = {10, 7, 8, 9, 1, 5};

        sort(arr, 0, arr.length - 1);

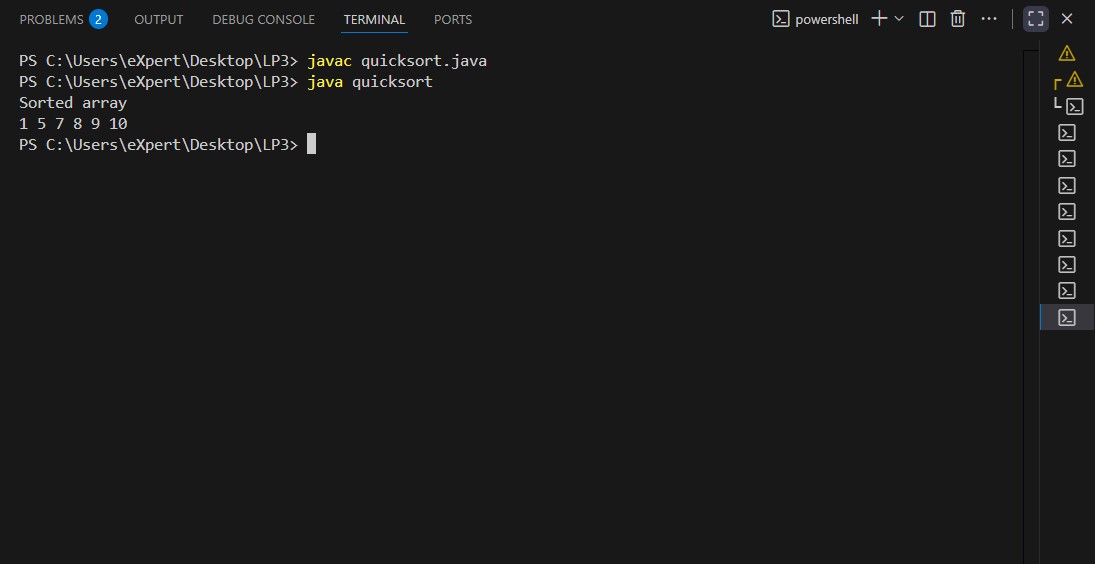
        System.out.println("Sorted array");

        printArray(arr);

    }

}

**Output:**

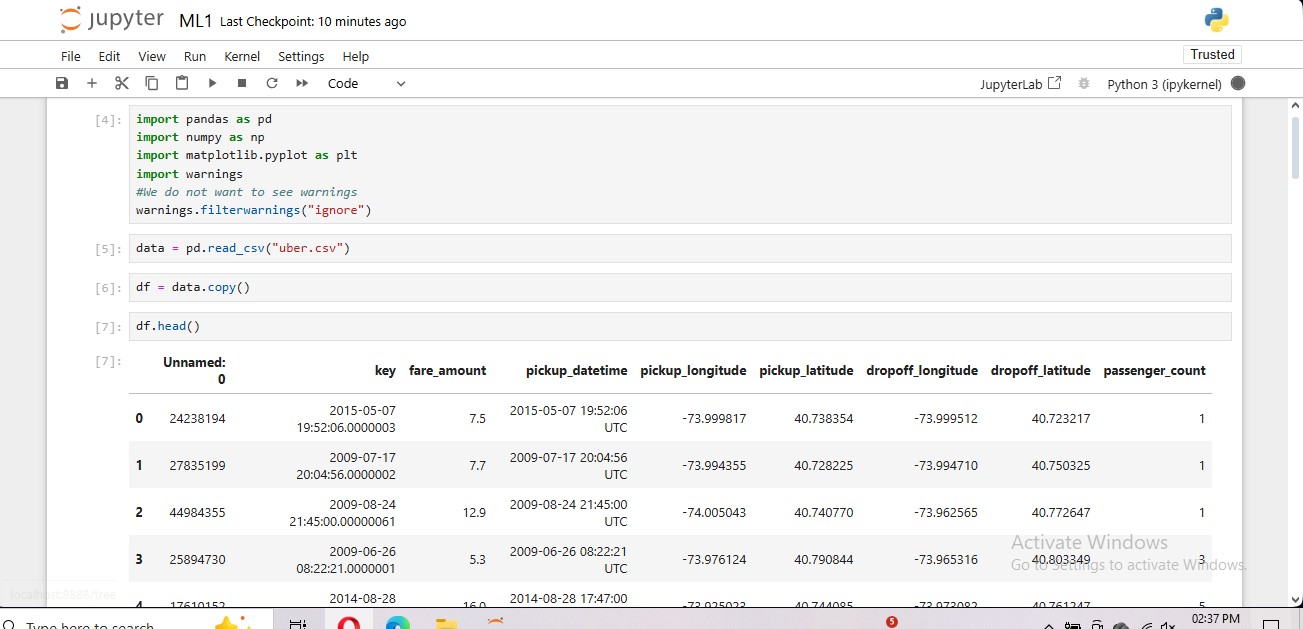
****

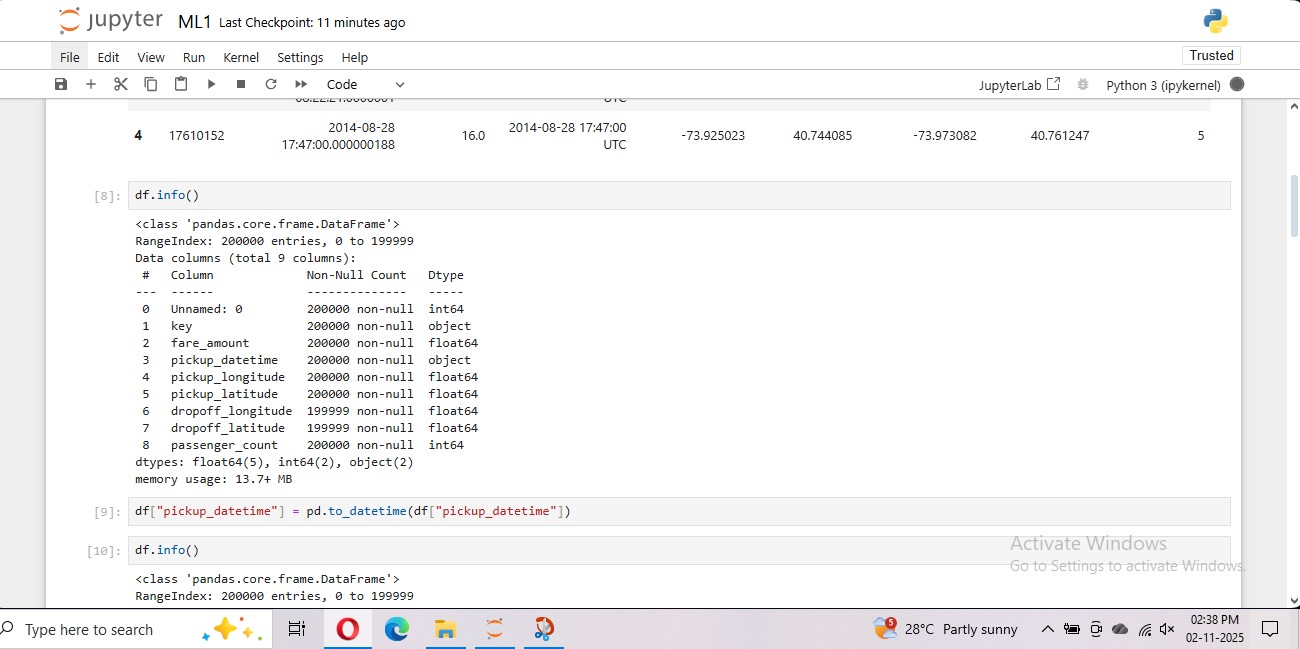
**ML**

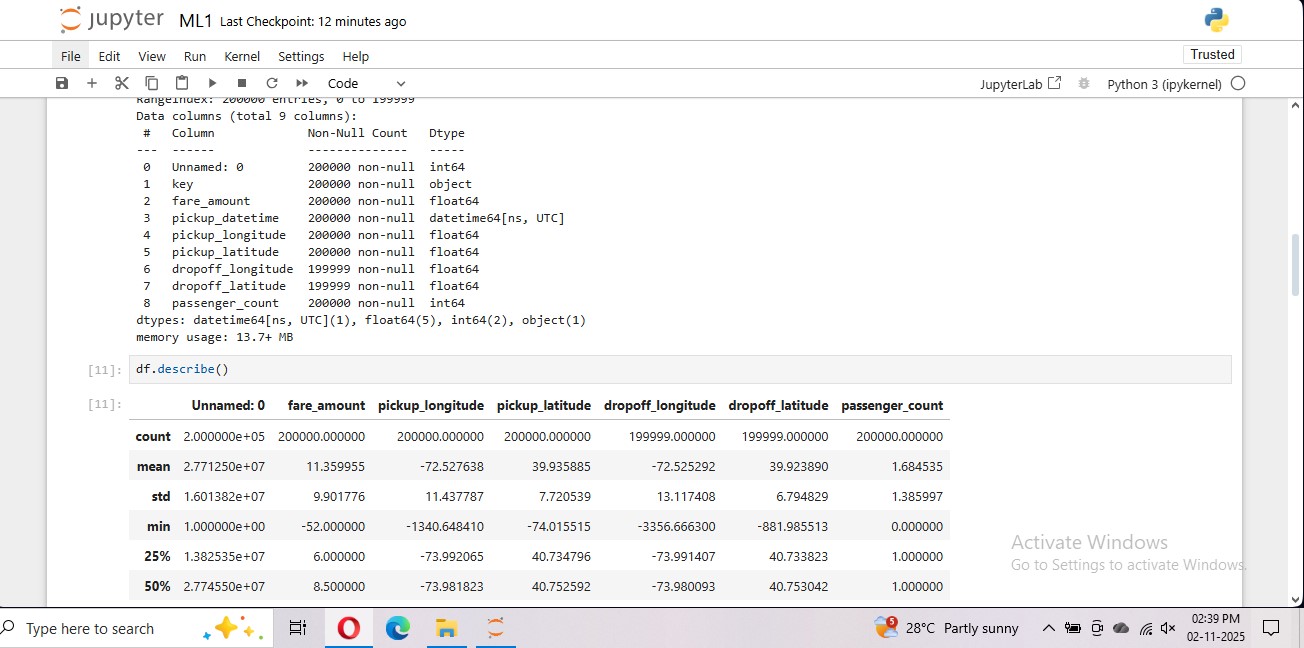
**ASSIGNMENT NO.1**

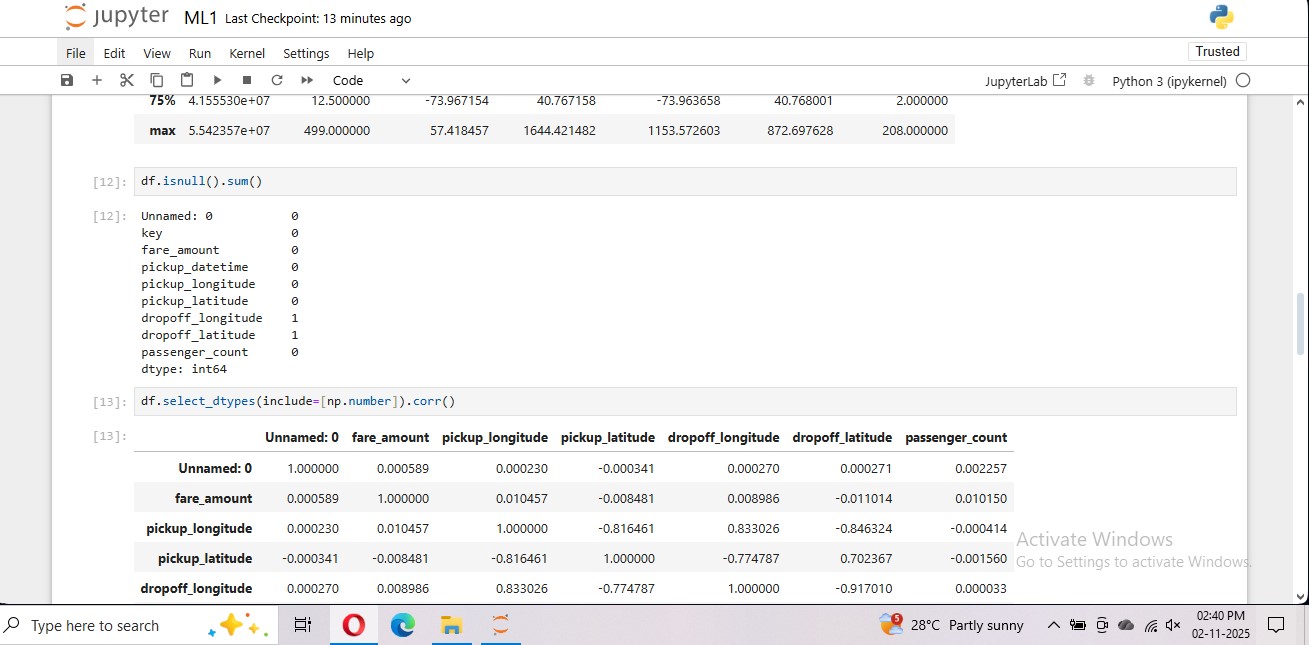
**Title:** Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

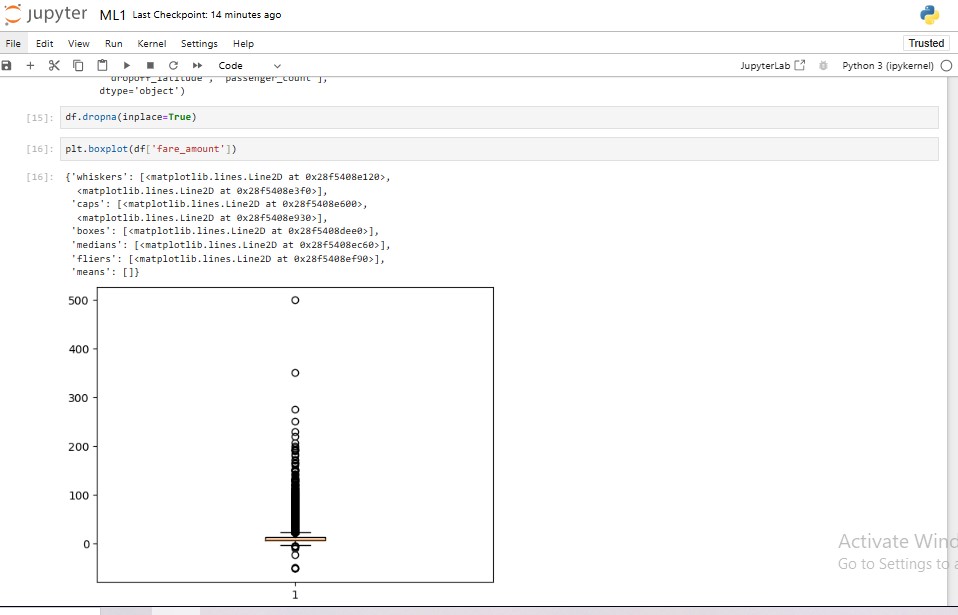
**Code/Output:**

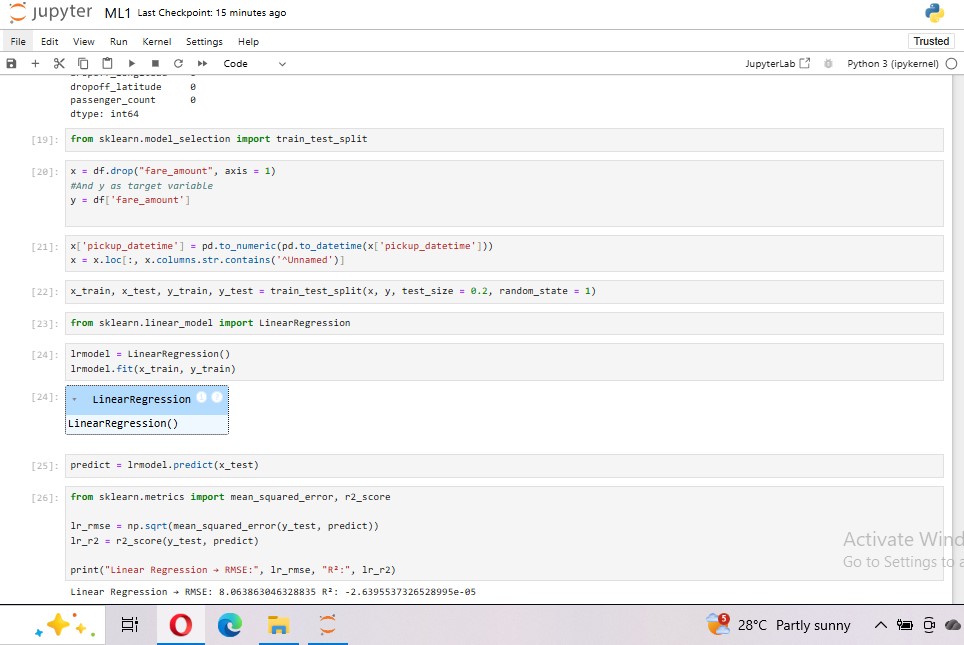
****

****

****

****

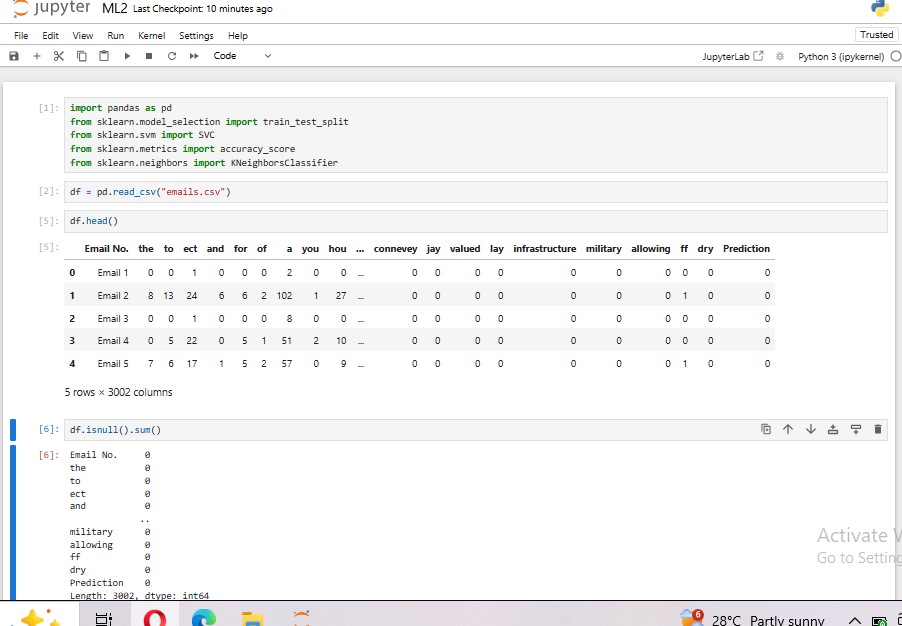
****

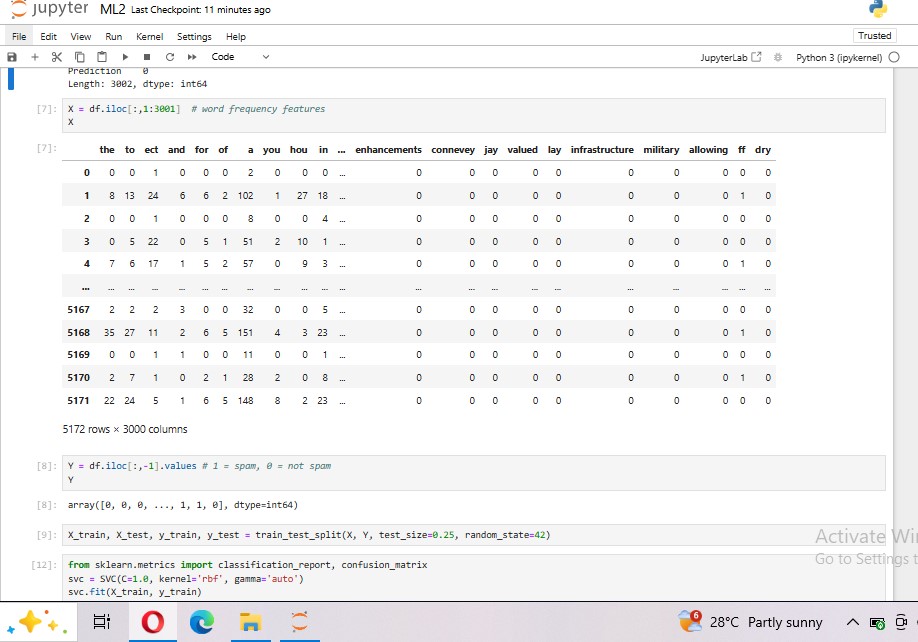
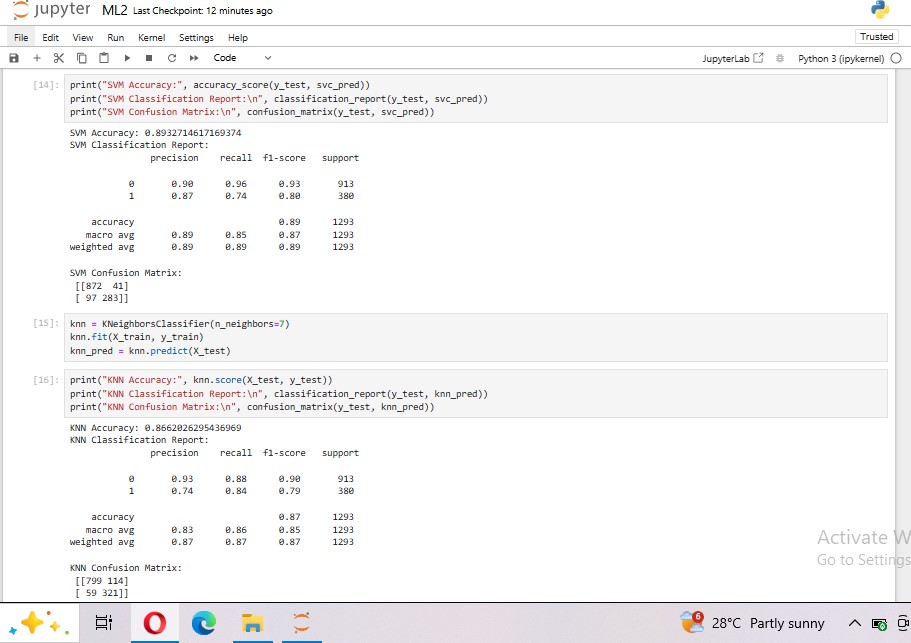
****

**ASSIGNMENT NO.2**

**Title**: Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

**Code\Output:**

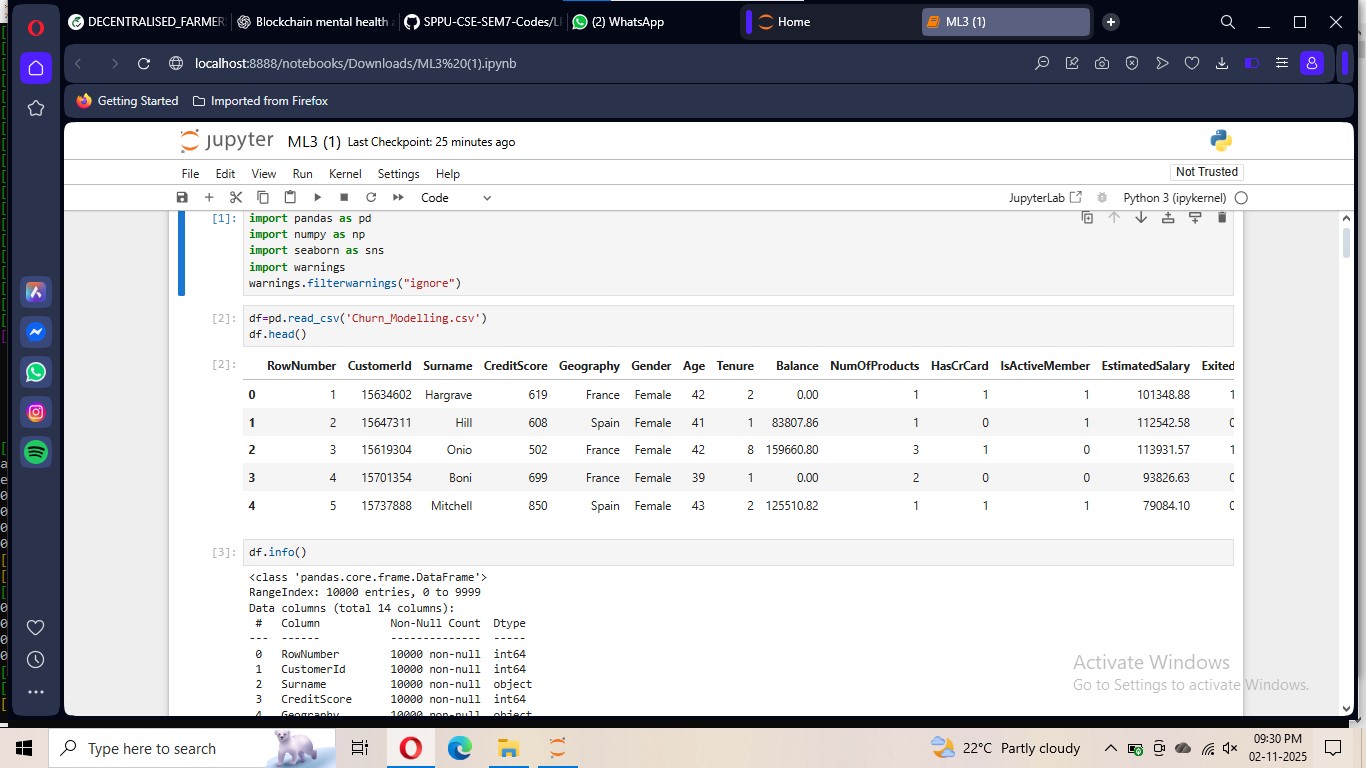
****

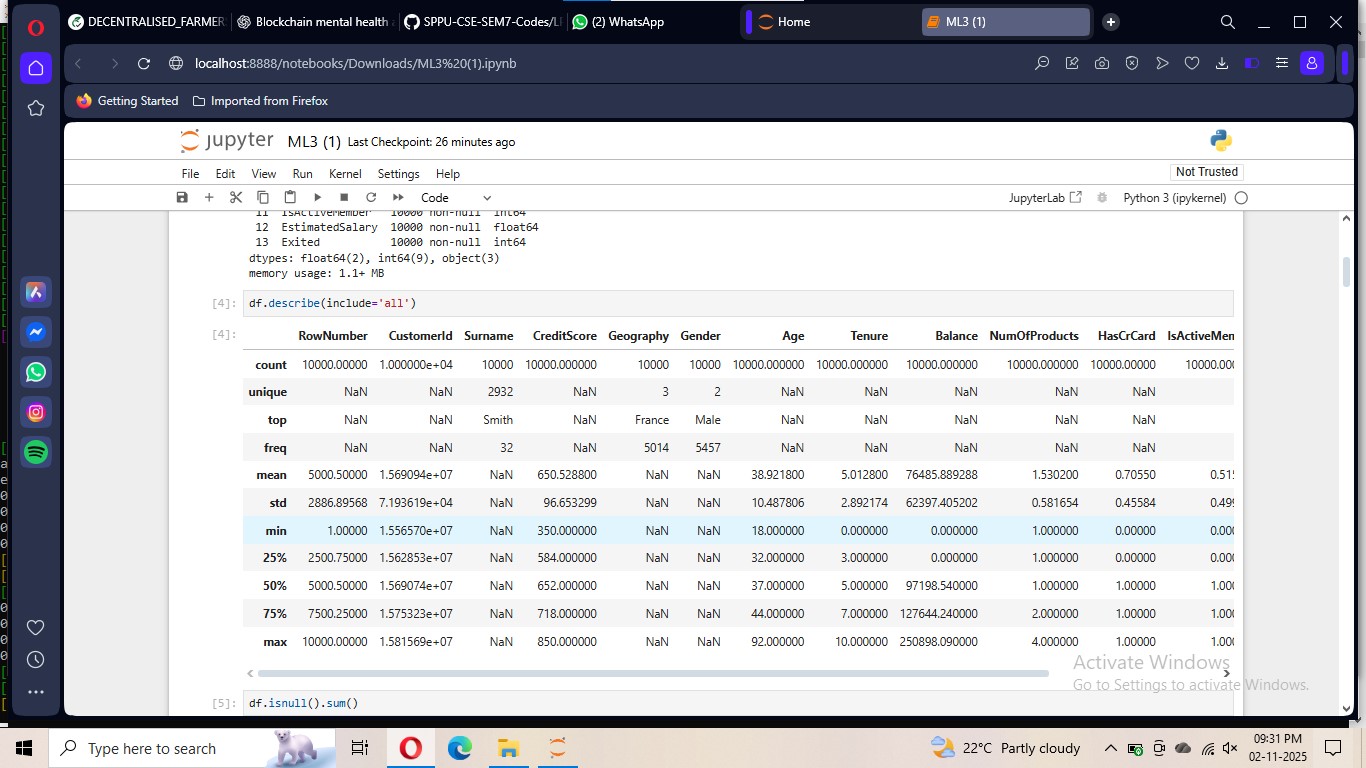
**  
**

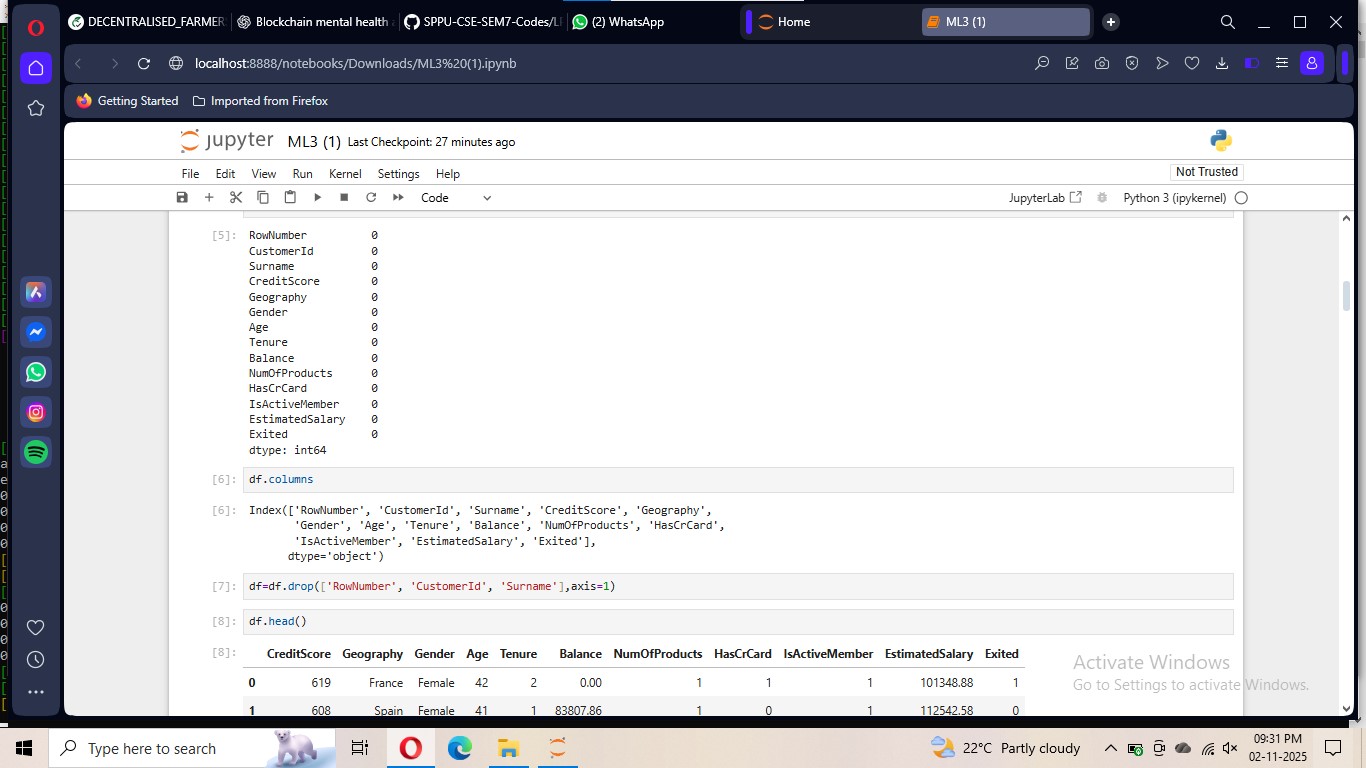
**ASSIGNMENT NO.3**

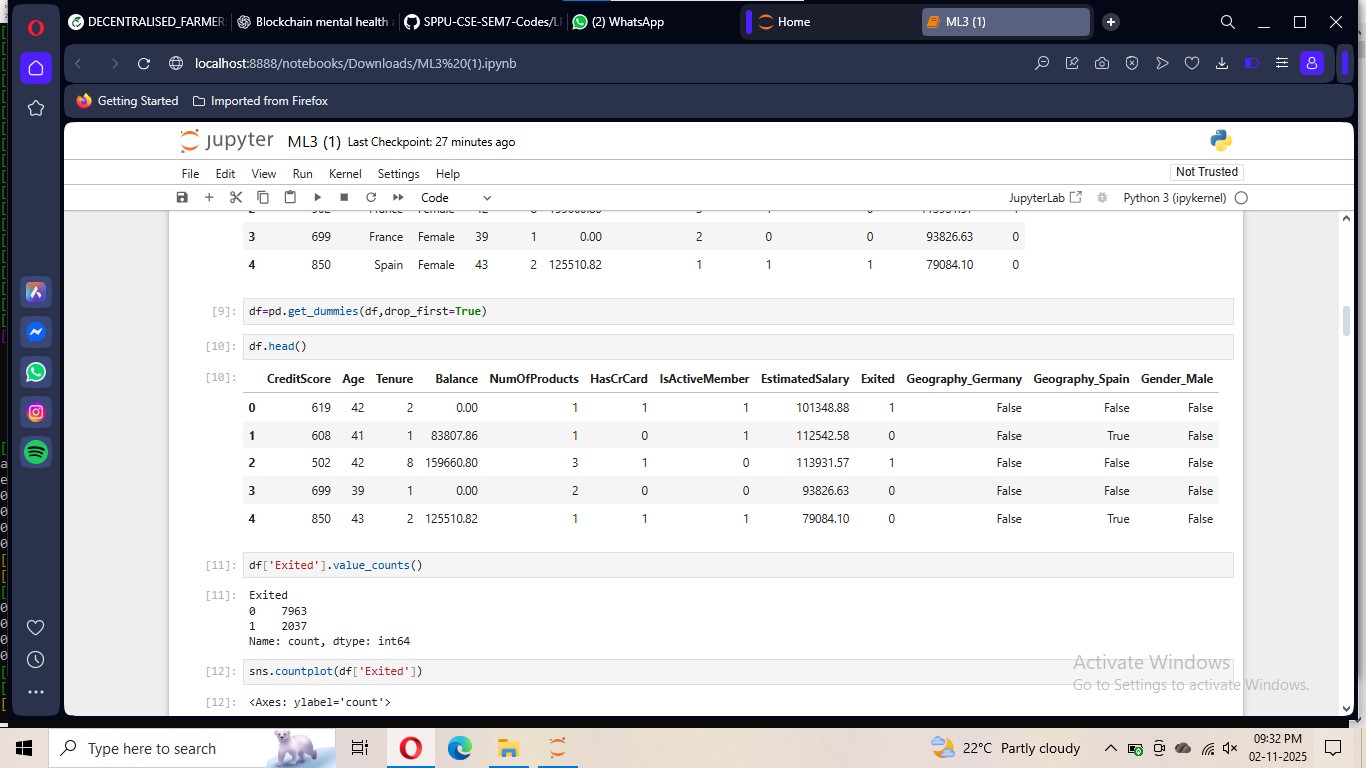
**Title:** Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

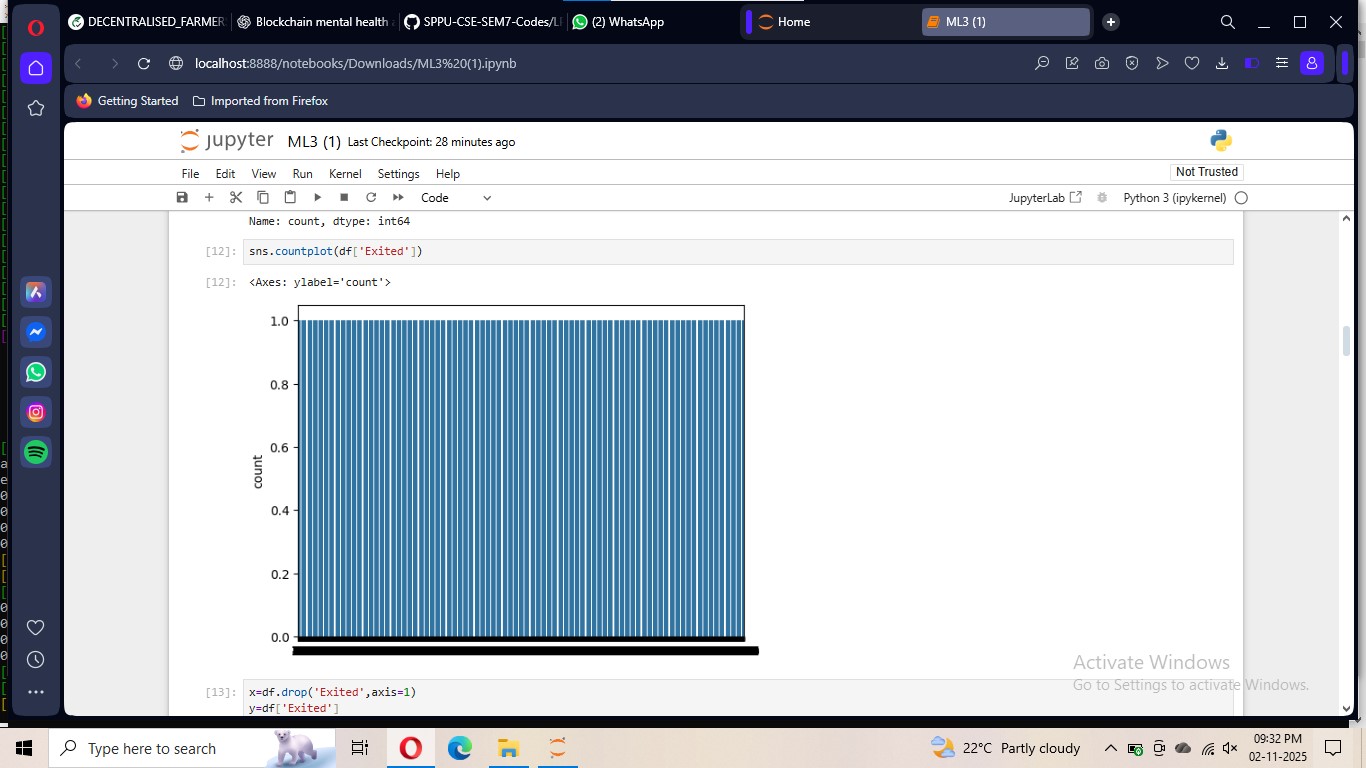
**Code\Output:**

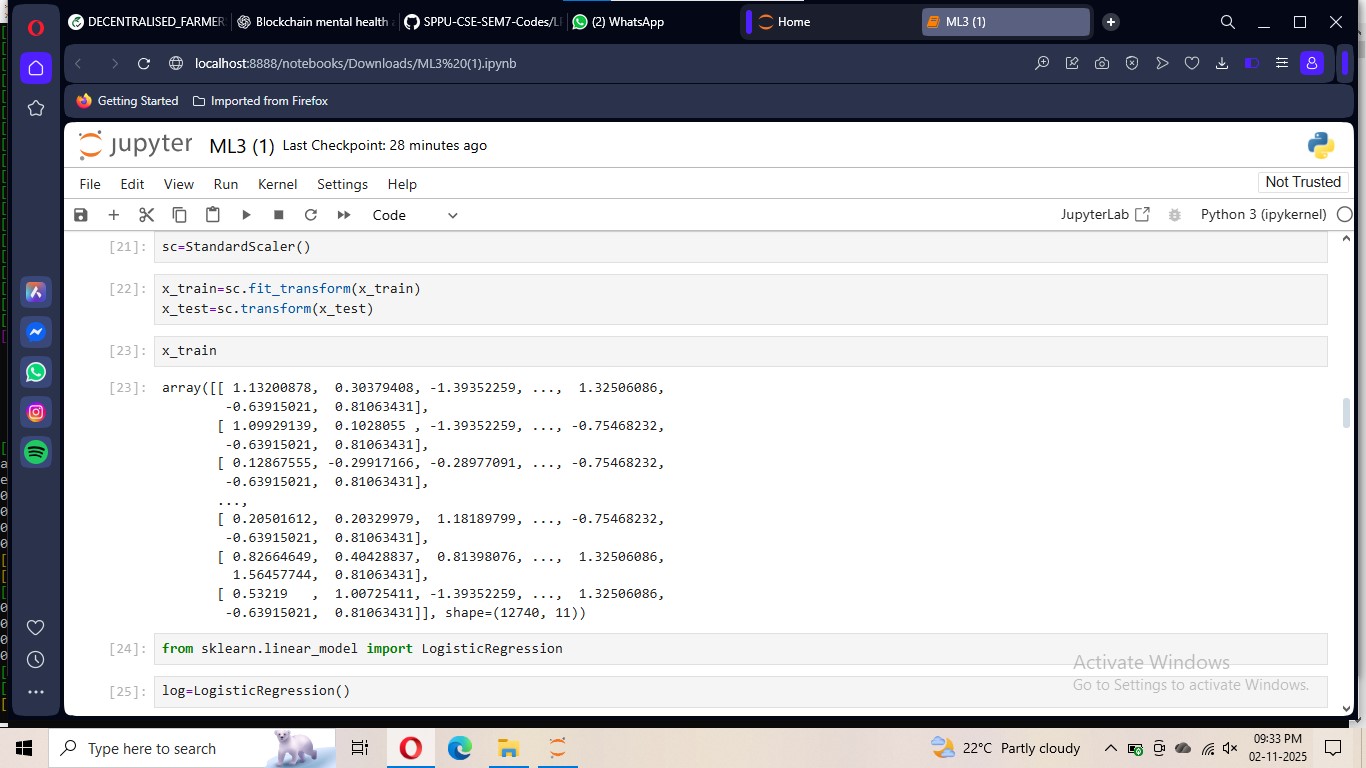
****

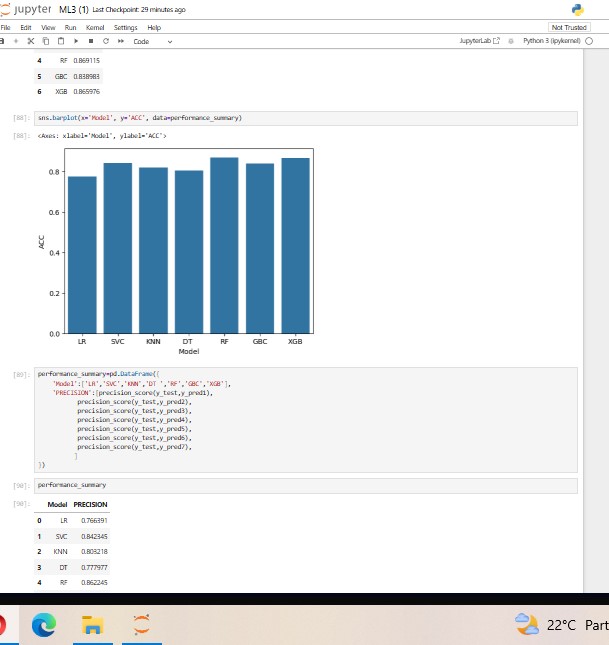


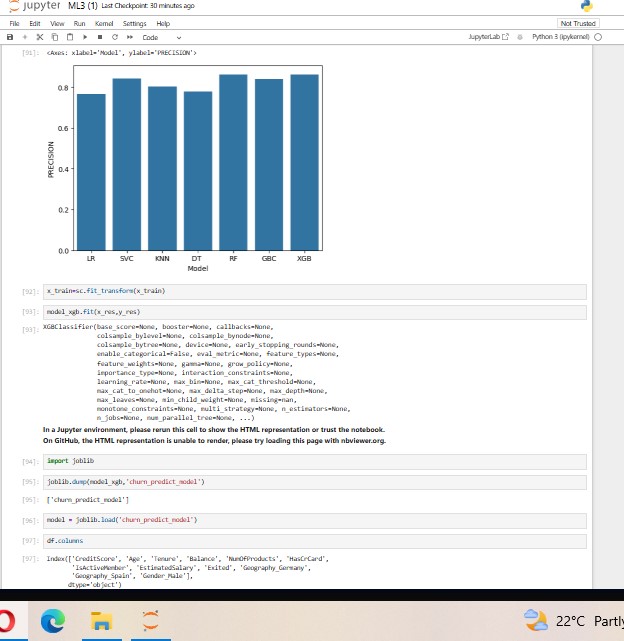




****

****

****

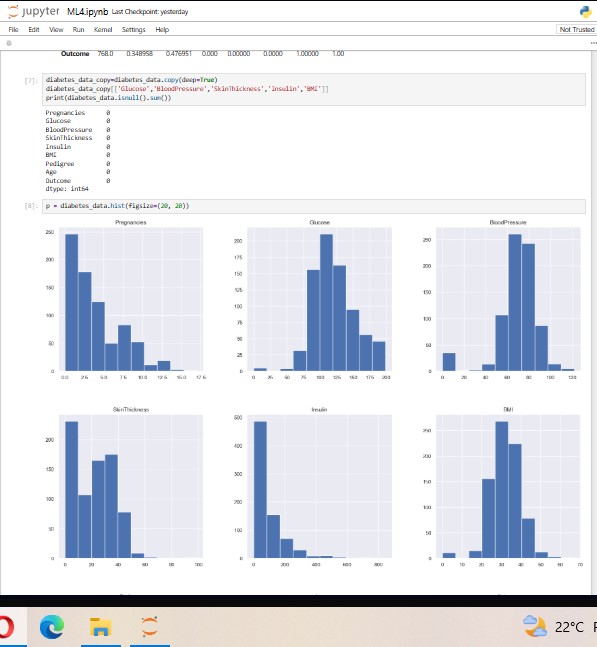
****

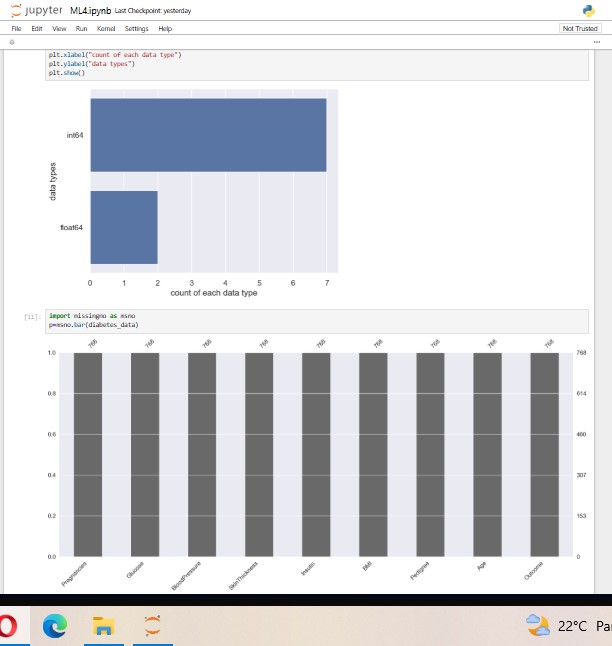
**ASSIGNMENT NO.4**

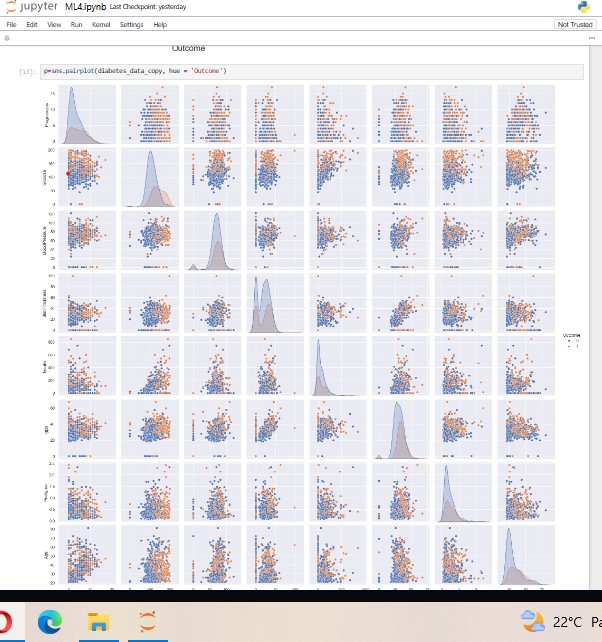
**Title:** Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

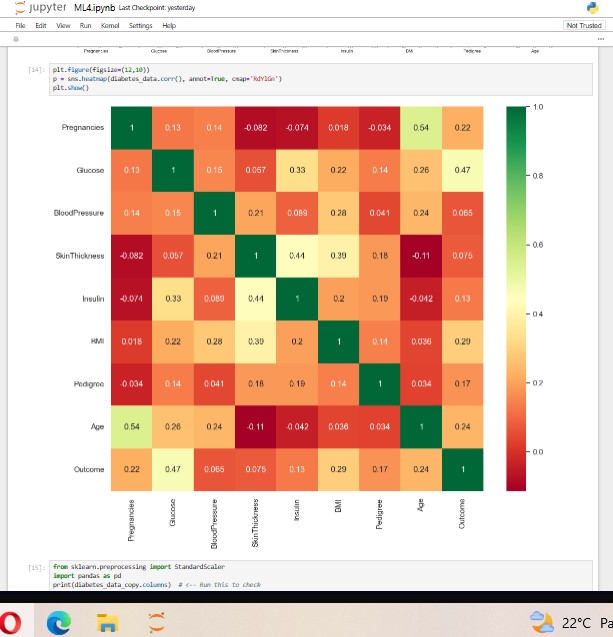
**Title\Output:**

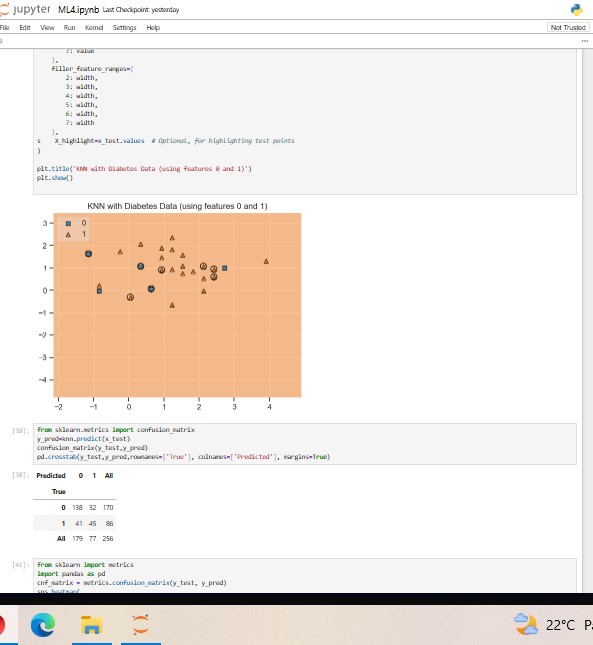


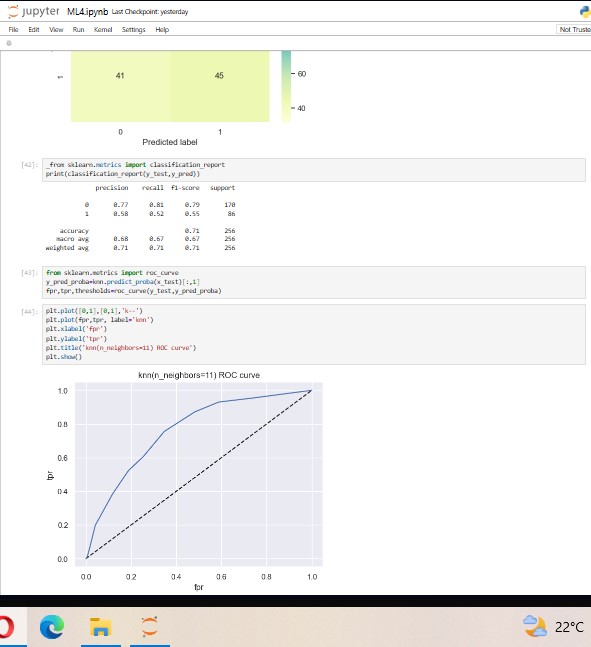
****

****

****

****

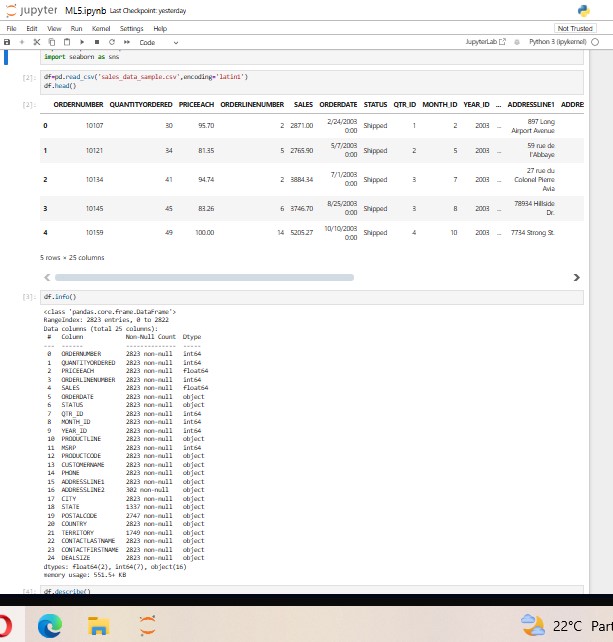
****

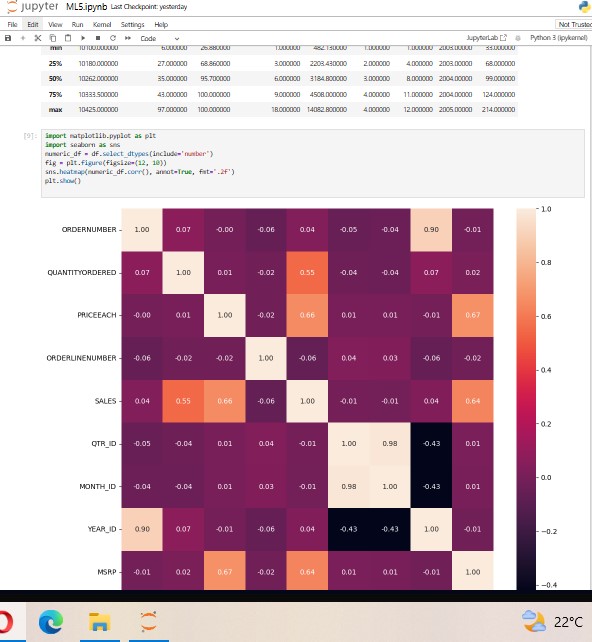
****

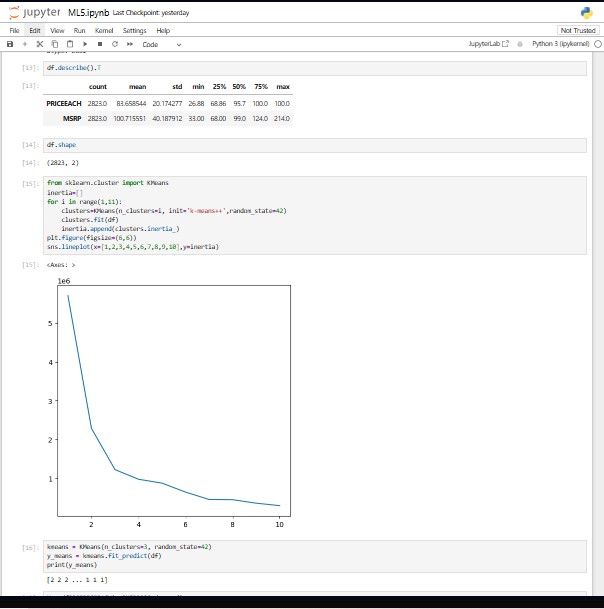
**ASSIGNMENT NO.5**

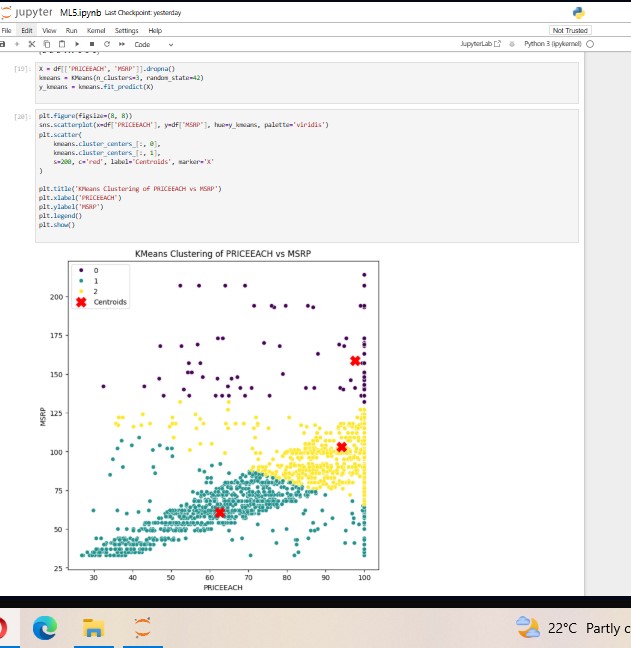
**Title:** Implement K-Means clustering/ hierarchical clustering on sales\_data\_sample.csv dataset. Determine the number of clusters using the elbow method.

**Code\Output:**







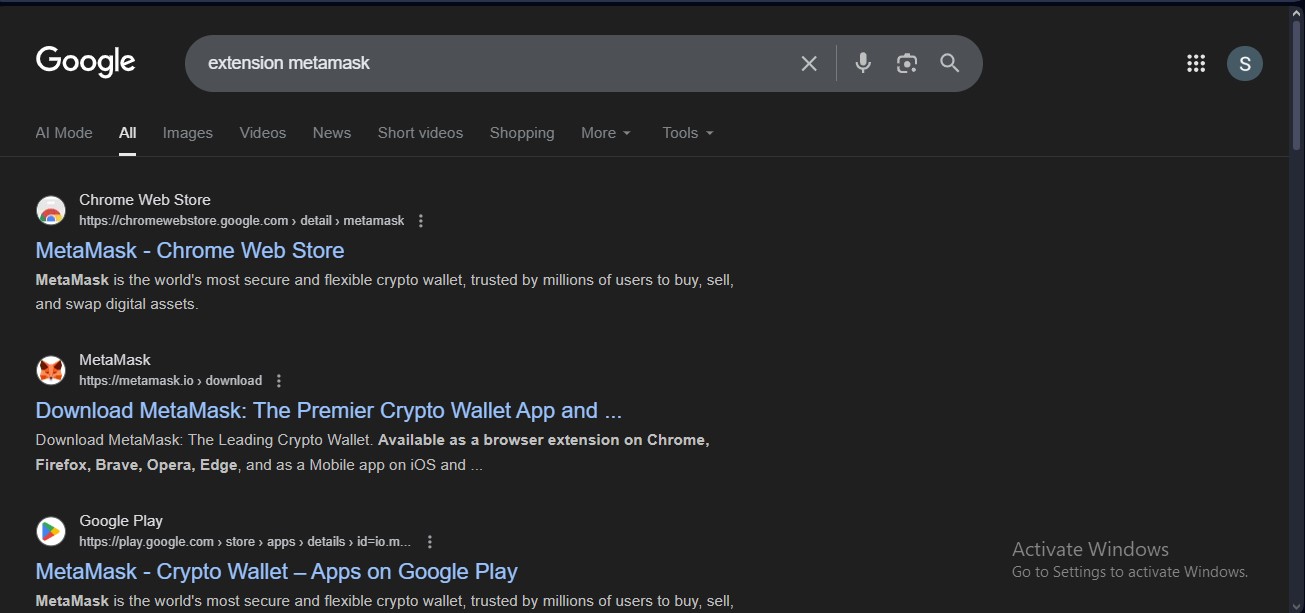


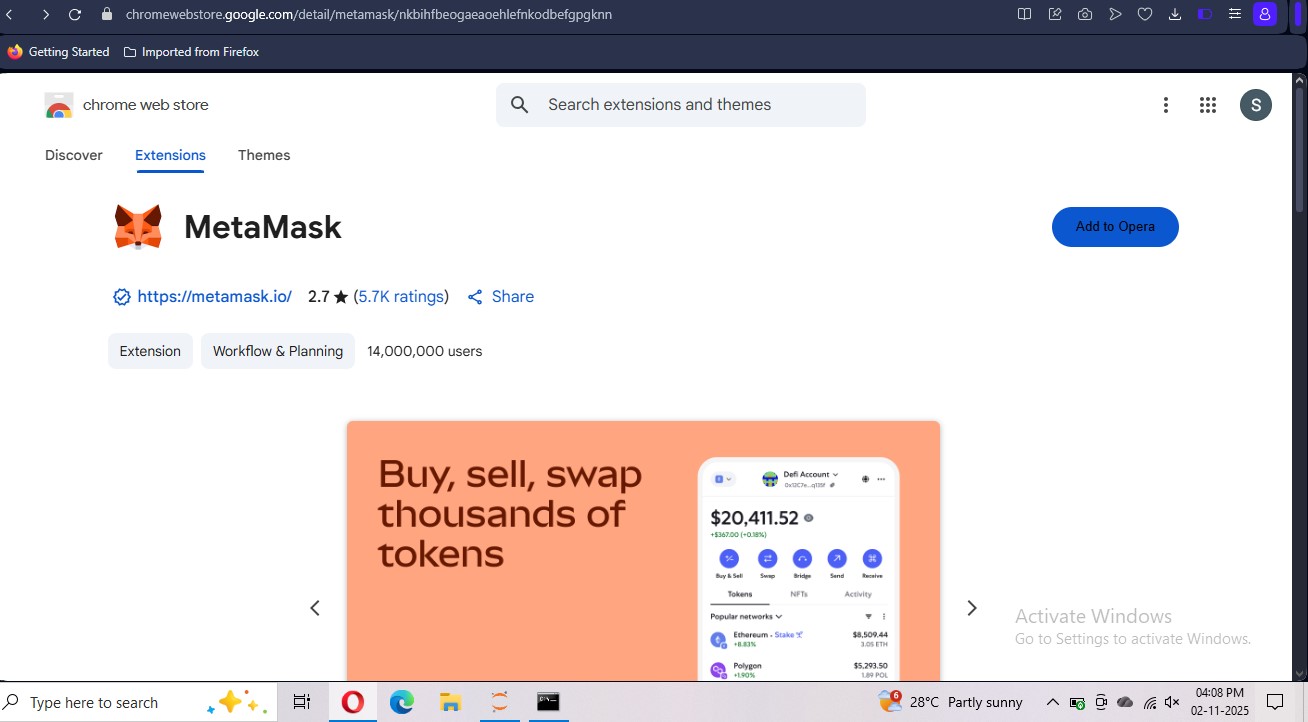
**BT**

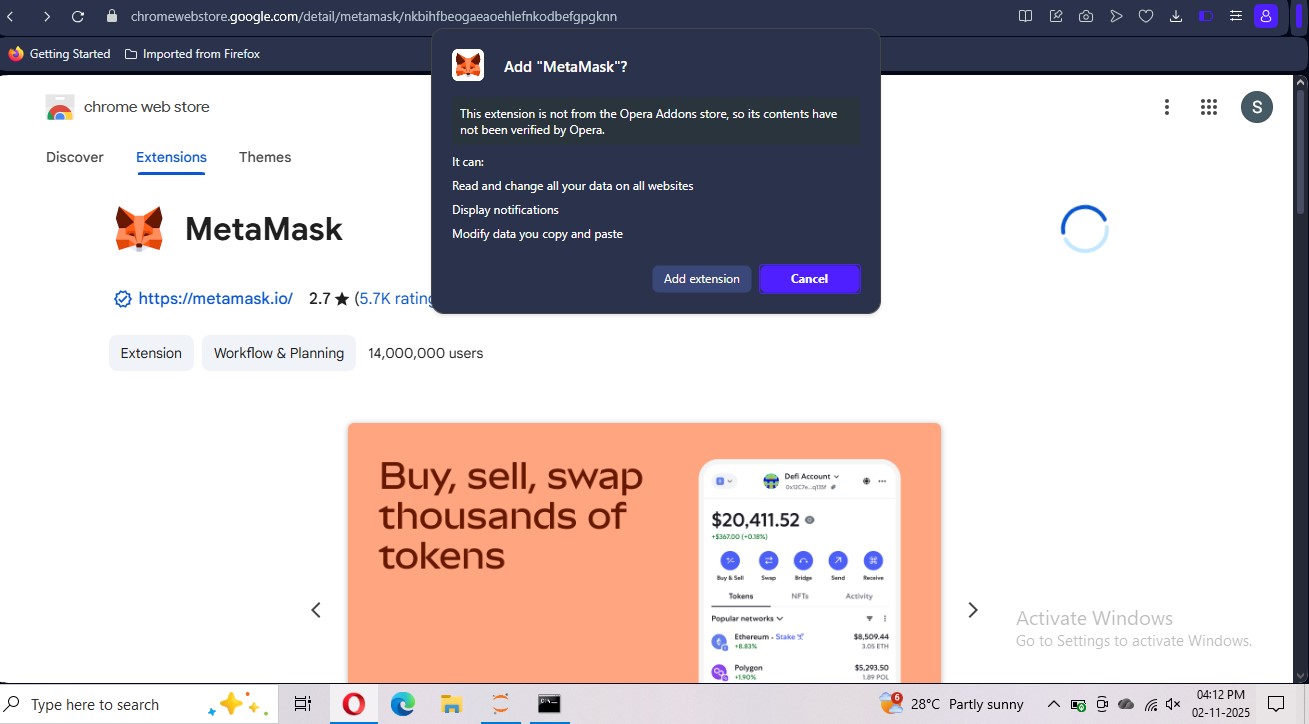
**ASSIGNMENT NO.1**

**Title:** Installation of Metamask and study spending Ether per transaction.

**Output:**

****



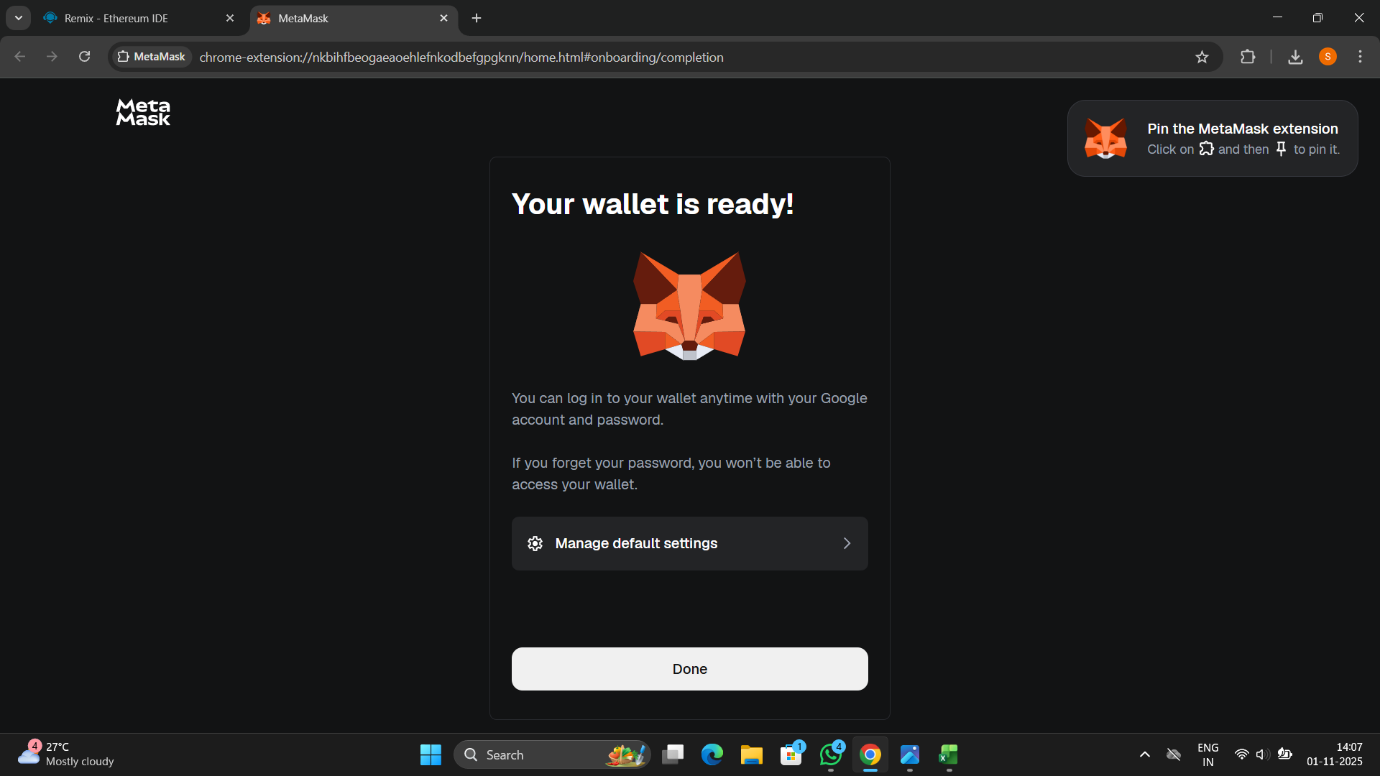


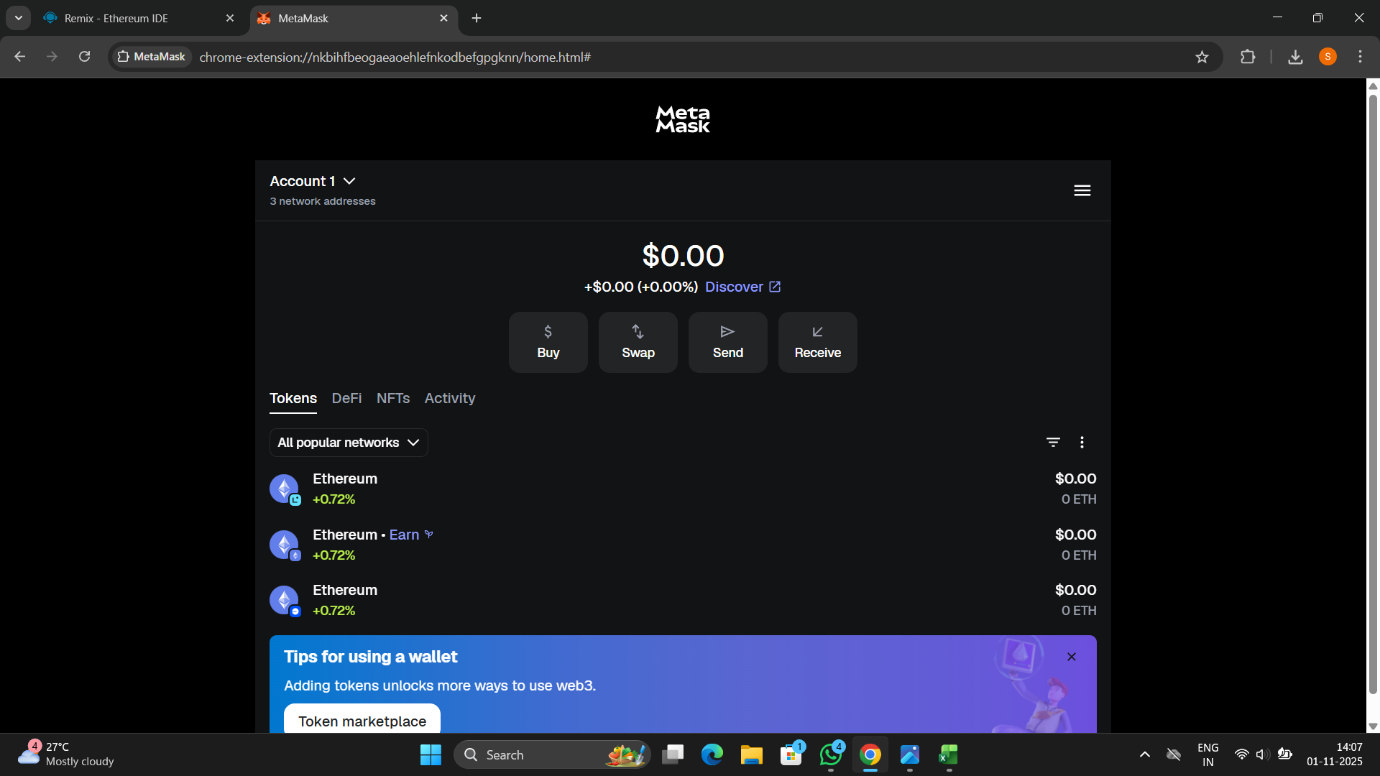
**ASSIGNMENT NO.2**

**Title:** Create your own wallet using Metamask for crypto transactions

**Output:**

****

****



**ASSIGNMENT NO.3**

**Title:** Write a smart contract on a test network, for Bank account of a customer for following operations:

**Code:**

**// SPDX-License-Identifier: UNLICENSED**

**pragma solidity >=0.8.0;**

**contract BankAccount {**

**mapping(address => uint256) private balances;**

**mapping(address => bool) private isUser;**

**event Deposit(address indexed user, uint256 amount);**

**event Withdraw(address indexed user, uint256 amount);**

**// Create account (optional initial deposit)**

**function createAccount() public payable {**

**require(!isUser[msg.sender], "Account already exists");**

**isUser[msg.sender] = true;**

**balances[msg.sender] = msg.value;**

**}**

**// Deposit money**

**function deposit() public payable {**

**require(isUser[msg.sender], "Account not found");**

**require(msg.value > 0, "Deposit must be > 0");**

**balances[msg.sender] += msg.value;**

**emit Deposit(msg.sender, msg.value);**

**}**

**// Withdraw money**

**function withdraw(uint256 amount) public {**

**require(isUser[msg.sender], "Account not found");**

**require(balances[msg.sender] >= amount, "Insufficient balance");**

**balances[msg.sender] -= amount;**

**(bool success, ) = msg.sender.call{value: amount}("");**

**require(success, "Withdrawal failed");**

**emit Withdraw(msg.sender, amount);**

**}**

**// Show your balance**

**function showBalance() public view returns (uint256) {**

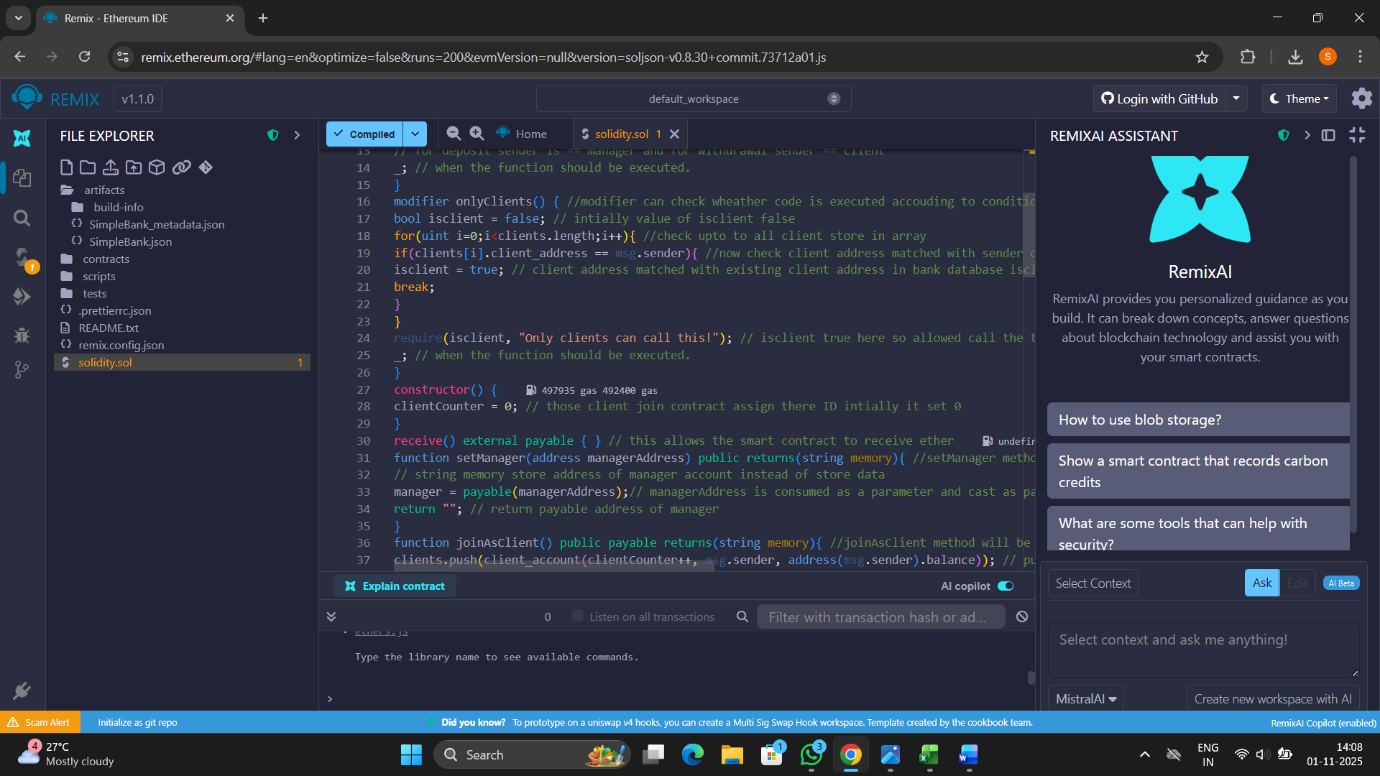
**require(isUser[msg.sender], "Account not found");**

**return balances[msg.sender];**

**}**

**}**

**Output:**

****

**Assignment No:4**

**Title:** Write a program in solidity to create Student data. Use the following constructs:

**Code:**

**// SPDX-License-Identifier: UNLICENSED**

**pragma solidity >=0.8.0;**

**contract StudentData {**

**// Structure to store student information**

**struct Student {**

**uint256 id;**

**string name;**

**uint8 age;**

**string course;**

**}**

**// Array to store multiple students**

**Student[] public students;**

**// Event for logging**

**event StudentAdded(uint256 id, string name);**

**// Function to add a new student**

**function addStudent(uint256 \_id, string memory \_name, uint8 \_age, string memory \_course) public {**

**Student memory newStudent = Student({**

**id: \_id,**

**name: \_name,**

**age: \_age,**

**course: \_course**

**});**

**students.push(newStudent);**

**emit StudentAdded(\_id, \_name);**

**}**

**// Function to get total number of students**

**function getStudentCount() public view returns (uint256) {**

**return students.length;**

**}**

**// Function to get a student by index**

**function getStudent(uint256 index) public view returns (uint256, string memory, uint8, string memory) {**

**require(index < students.length, "Invalid index");**

**Student storage s = students[index];**

**return (s.id, s.name, s.age, s.course);**

**}**

**// Fallback function to accept ether**

**fallback() external payable {**

**// Do nothing, just accept ETH**

**}**

**receive() external payable {**

**// Optional: handle plain ETH transfers**

**}**

**// Function to get contract balance**

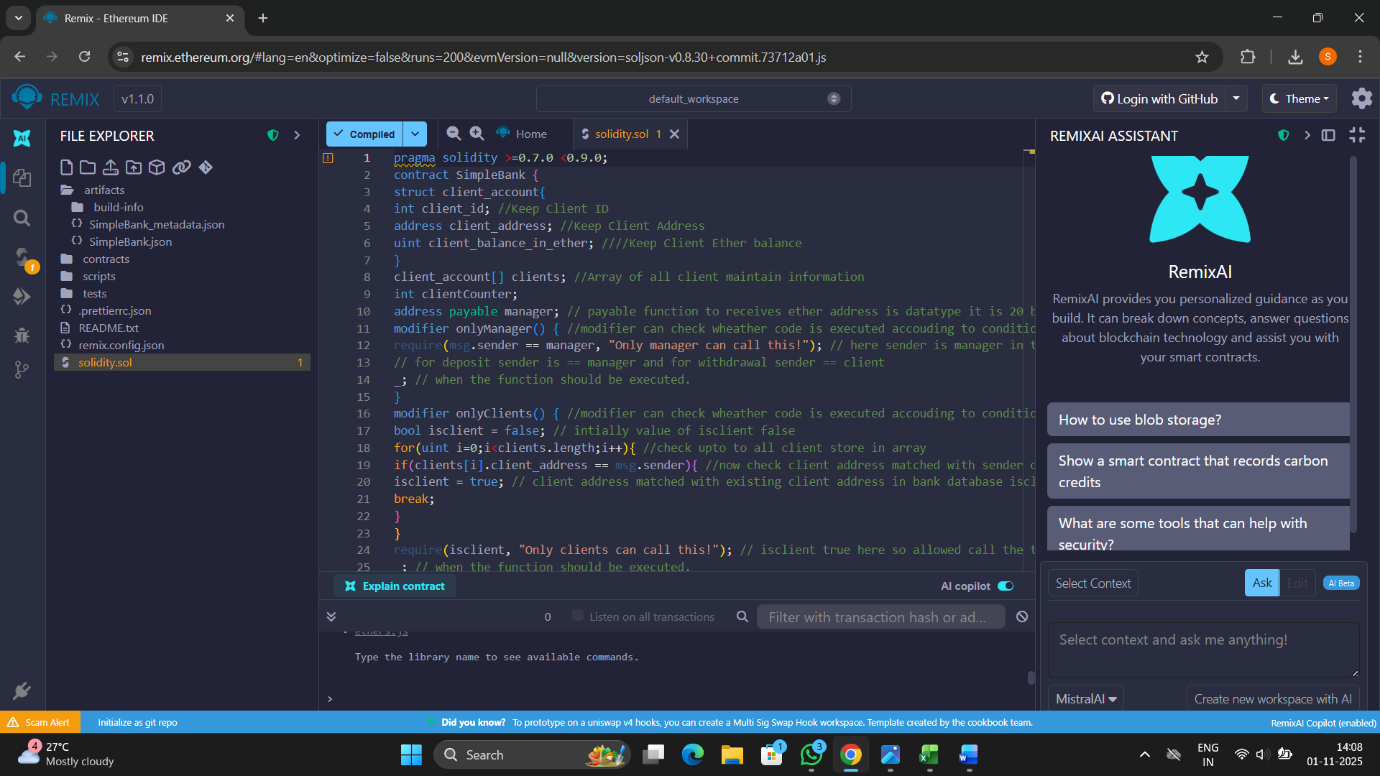
**function getContractBalance() public view returns (uint256) {**

**return address(this).balance;**

**}**

**}**

**Output:**

****

**Assignment No:5**

**Title:** Write a survey report on types of Blockchains and its real time use cases.

**Code:**

**// SPDX-License-Identifier: UNLICENSED**

**pragma solidity >=0.8.0;**

**contract StudentData {**

**// Structure to store student information**

**struct Student {**

**uint256 id;**

**string name;**

**uint8 age;**

**string course;**

**}**

**// Array to store multiple students**

**Student[] public students;**

**// Event for logging**

**event StudentAdded(uint256 id, string name);**

**// Function to add a new student**

**function addStudent(uint256 \_id, string memory \_name, uint8 \_age, string memory \_course) public {**

**Student memory newStudent = Student({**

**id: \_id,**

**name: \_name,**

**age: \_age,**

**course: \_course**

**});**

**students.push(newStudent);**

**emit StudentAdded(\_id, \_name);**

**}**

**// Function to get total number of students**

**function getStudentCount() public view returns (uint256) {**

**return students.length;**

**}**

**// Function to get a student by index**

**function getStudent(uint256 index) public view returns (uint256, string memory, uint8, string memory) {**

**require(index < students.length, "Invalid index");**

**Student storage s = students[index];**

**return (s.id, s.name, s.age, s.course);**

**}**

**// Fallback function to accept ether**

**fallback() external payable {**

**// Do nothing, just accept ETH**

**}**

**receive() external payable {**

**// Optional: handle plain ETH transfers**

**}**

**// Function to get contract balance**

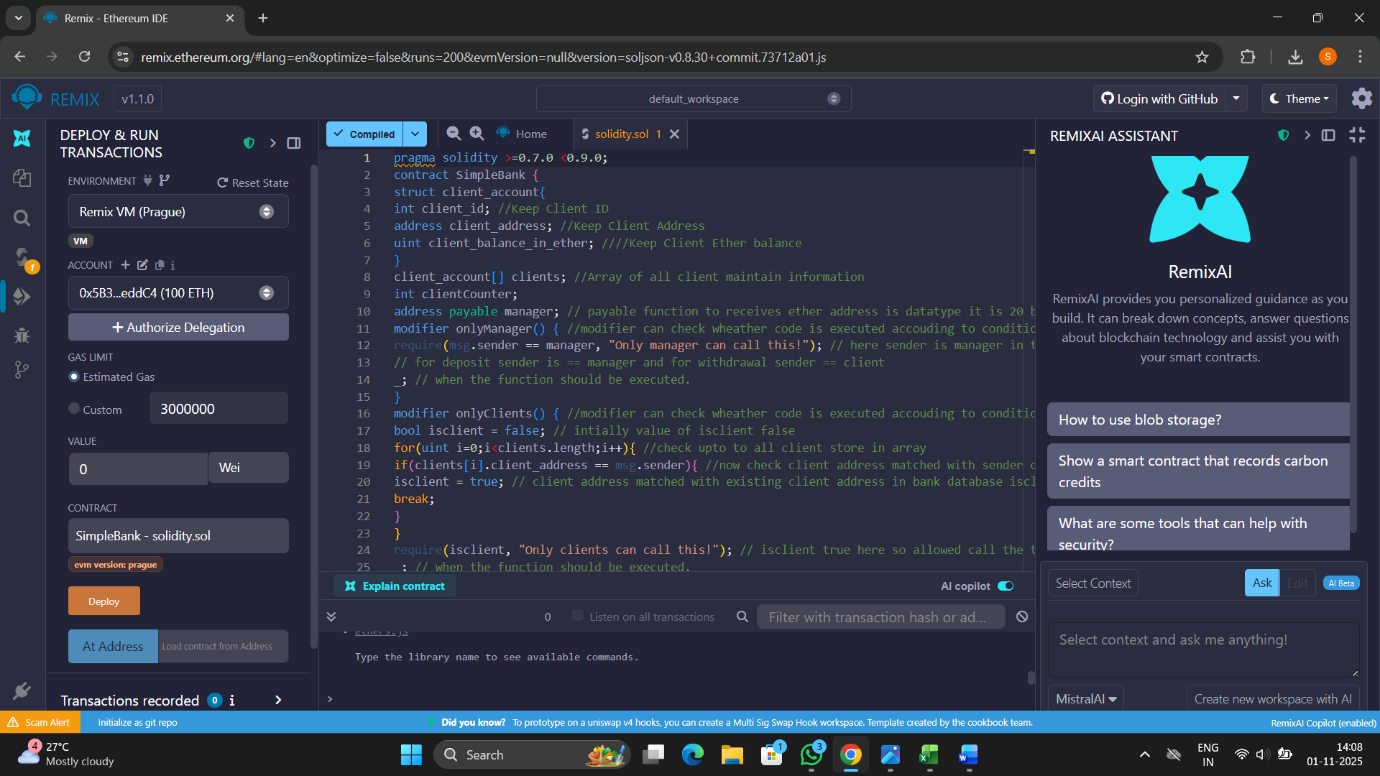
**function getContractBalance() public view returns (uint256) {**

**return address(this).balance;**

**}**

**}**

**Output:**

****