|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **Course Coordinator Name** | | | | Dr. Rishabh Mittal | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Mr. S Naresh Kumar | | Ms. B. Swathi | | Dr. Sasanko Shekhar Gantayat | | Mr. Md Sallauddin | | Dr. Mathivanan | | Mr. Y Srikanth | | Ms. N Shilpa | | Dr. Rishabh Mittal (Coordinator) | | Dr. R. Prashant Kumar | | Mr. Ankushavali MD | | Mr. B Viswanath | | Ms. Sujitha Reddy | | Ms. A. Anitha | | Ms. M.Madhuri | | Ms. Katherashala Swetha | | Ms. Velpula sumalatha | | Mr. Bingi Raju | | | | | | |
| **CourseCode** | | | 23CS002PC304 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | III/II | **Regulation** | | R23 | | | |
| **Date and Day**  **of Assignment** | | | **Week1 – Wednesday** | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All batches | | | |
| **Assignment Number:1.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI  **Lab Objectives:**   * To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. * To understand and use Cursor AI for code generation, explanation, and refactoring. * To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. * To perform code optimization and documentation using AI tools.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Generate Python code using Google Gemini in Google Colab. * Analyze the effectiveness of code explanations and suggestions by Gemini. * Set up and use Cursor AI for AI-powered coding assistance. * Evaluate and refactor code using Cursor AI features. * Compare AI tool behavior and code quality across different platforms.   **Task 1: Word Frequency from Text File**   * **Scenario:** You are analyzing log files for keyword frequency. * **Task:** Use Gemini to generate Python code that reads a text file and counts word frequency, then explains the code. * **Expected Output:**   + Working code   + Explanation   + Screenshot   **Prompt:-**    **Code:-**  Created a text file in Gemini colab      **Output:-**    **Explanation:-**  **What we did:**   1. We created a sample text file named 'sample\_text.txt' with some content. 2. We then ran Python code that reads this file.   3.The code processed the text, counted the frequency of each word, and printed the results  **What the code does:**This code effectively analyzes a given text file to determine how often each word appears. It treats words like 'apple' and 'Apple' as the same by converting everything to lowercase. It then lists these words along with their counts, ordered from the most common to the least common, providing insights into the most frequent terms in the document.  **Task 2: File Operations Using Cursor AI**   * **Scenario:** You are automating basic file operations. * **Task:** Use Cursor AI to generate a program that:   + Creates a text file   + Writes sample text   + Reads and displays the content * **Expected Output:**   + Functional code   + Cursor AI screenshots   **Prompt:-**      **Code:-**    **Output:-**  Hello! This is a sample text file.  This file is created using Python.  File operations are easy to learn.  **Explanation:-**  In this task, basic file operations were automated using Cursor AI to understand how Python handles file handling. The program was designed to create a text file, write sample content into it, and then read and display the same content on the screen. This task helps beginners learn how data can be stored permanently using files instead of temporary memory. It also demonstrates the use of file modes such as write ("w") and read ("r"), along with proper opening and closing of files. By using Cursor AI, the process of generating clean and well-commented code became easier and more efficient. Overall, this task improves understanding of real-world applications like data storage, report generation, and logging using Python file operations.  **Task 3: CSV Data Analysis**   * **Scenario:** You are processing structured data from a CSV file. * **Task:** Use Gemini in Colab to read a CSV file and calculate mean, min, and max. * **Expected Output:**   + Correct output   + Screenshot   **Prompt:-**    **Code:-**      **Output:-**    **Explanation:-**  This Python code uses the pandas library to efficiently analyze data from a CSV file. It begins by reading the 'sample\_data.csv' file into a pandas DataFrame, which is a structured way to hold tabular data. Once loaded, the code accesses the 'Value' column of this DataFrame and then calculates three key statistical measures: the mean (average), the minimum value, and the maximum value. Finally, these calculated statistics are printed to the console, providing a quick summary of the numerical data in the 'Value' column.  **Task 4: Sorting Lists – Manual vs Built-in**   * **Scenario:** You are reviewing algorithm choices for efficiency. * **Task:** Use **Gemini** to generate:   + Bubble sort   + Python’s built-in sort()   + Compare both implementations. * **Expected Output:**   + Two versions of code   + Short comparison   **Prompt:-**    **Code & output:-**      **Explanation:-**  **Comparison of Bubble Sort vs. Python's Built-in sort()**  Bubble Sort is a simple comparison-based algorithm with a time complexity of O(n²) in the worst and average cases, making it highly inefficient for large datasets. Its primary use is educational, illustrating basic sorting principles. In contrast, Python's built-in list.sort() method is an optimized Timsort algorithm, which is a hybrid stable sorting algorithm with an average and worst-case time complexity of O(n log n). Timsort is significantly faster and more efficient, especially for larger lists, and is the preferred choice for practical applications due to its performance, stability, and ease of use, as it requires only a single function call.  **Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.** | | | | | | Week1 - Monday |  |