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:**	Week1 - Monday

- 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:**
  - Prompts

```
Write a Python code to count word frequency from a
text file and handling case insensitivity and
punctuation give me the optimized code.
```

  - Working code

```
"""Count word frequencies in a log file."""

import argparse
from collections import Counter
from pathlib import Path
```

```
def word_frequency(file_path: Path) ->
Counter[str]:
    text = file_path.read_text(encoding="utf-8")
    return Counter(text.lower().split())


def main(argv=None) -> int:
    parser = argparse.ArgumentParser(
        description="Count word frequencies in a
log file.",
        epilog="Run without arguments to process
logfile.txt in the current directory\nor pass a
custom path such as logs/server.log.",
    )
    parser.add_argument(
        "logfile",
        nargs="?",
        default="logfile.txt",
        help="Path to the log file to analyze.",
    )
    args = parser.parse_args(argv)
    file_path = Path(args.logfile)
    if not file_path.exists():
        parser.error(f"{file_path} does not exist.
Create that file or provide another path.")

    frequency = word_frequency(file_path)
    for word, count in frequency.most_common():
        print(f"{word}: {count}")
    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

- Explanation

```
You give it a text file.
The program reads the whole file.
It changes all the capital letters to small
letters.
It breaks the text apart so it has a list of every
single word.
It then uses the `Counter` class from the
`collections` module to count the frequency
Finally, it prints out a list that shows every
word and its total count.
```

- Screenshot

```
PS C:\Users\mahes\OneDrive\Desktop\HPC> & C:\Users\mahes\AppData\Local\Programs\Python\Pyth
on.exe c:/Users/mahes/OneDrive/Desktop/HPC/Task1.py
2026-01-08: 5
info: 3
task: 3
a: 2
10:00:00: 1
starting: 1
batch: 1
job: 1
warn: 1
10:05:12: 1
retry: 1
attempt: 1
1: 1
for: 1
10:07:30: 1
completed: 1
successfully: 1
error: 1
10:12:45: 1
b: 1
failed:: 1
timeout: 1
10:15:00: 1
cleanup: 1
```

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:**

- Prompts

```
Write a Python code to creates a file named
'sample.txt', writes three lines of example text
to it, then reads the file back and prints the
content to the console.
```

- Functional code

```python
def main() -> None:
    file_name = 'sample.txt'

    # Writing to the file
    with open(file_name, 'w') as file:
        file.write("This is the first line of
example text.\n")
        file.write("This is the second line of
example text.\n")
        file.write("This is the third line of
example text.\n")

    # Reading from the file
    with open(file_name, 'r') as file:
        content = file.read()

    # Printing the content to the console
    print("Content of 'sample.txt':")
    print(content)
if __name__ == "__main__":
    main()
```

- Cursor AI screenshots

```
PS C:\Users\mahes\AAC> & C:\Users\mahes\AppData\Local\Programs\Python\Python313\python.exe
ahes/AAC/Task2.py
Content of 'sample.txt':
This is the first line of example text.
This is the second line of example text.
This is the third line of example text.
```

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:**

  - Prompt

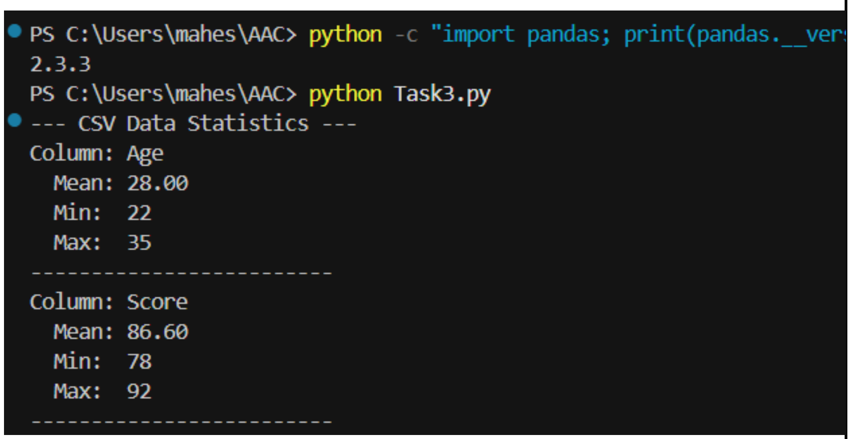
```
Write a Python code that read a CSV file and
calculates the mean, min, and max of a specific
numerical column.
```

  - Functional code

```
import csv
def calculate_statistics(file_name: str,
column_name: str) -> None:
    values = []

    # Reading the CSV file
    with open(file_name, mode='r') as csvfile:
        csvreader = csv.DictReader(csvfile)
        for row in csvreader:
            try:
                value = float(row[column_name])
                values.append(value)
            except ValueError:
                continue  # Skip rows where
conversion fails

    if not values:
        print(f"No valid data found in column
'{column_name}'.")
        return

    mean_value = sum(values) / len(values)
    min_value = min(values)
    max_value = max(values)
```

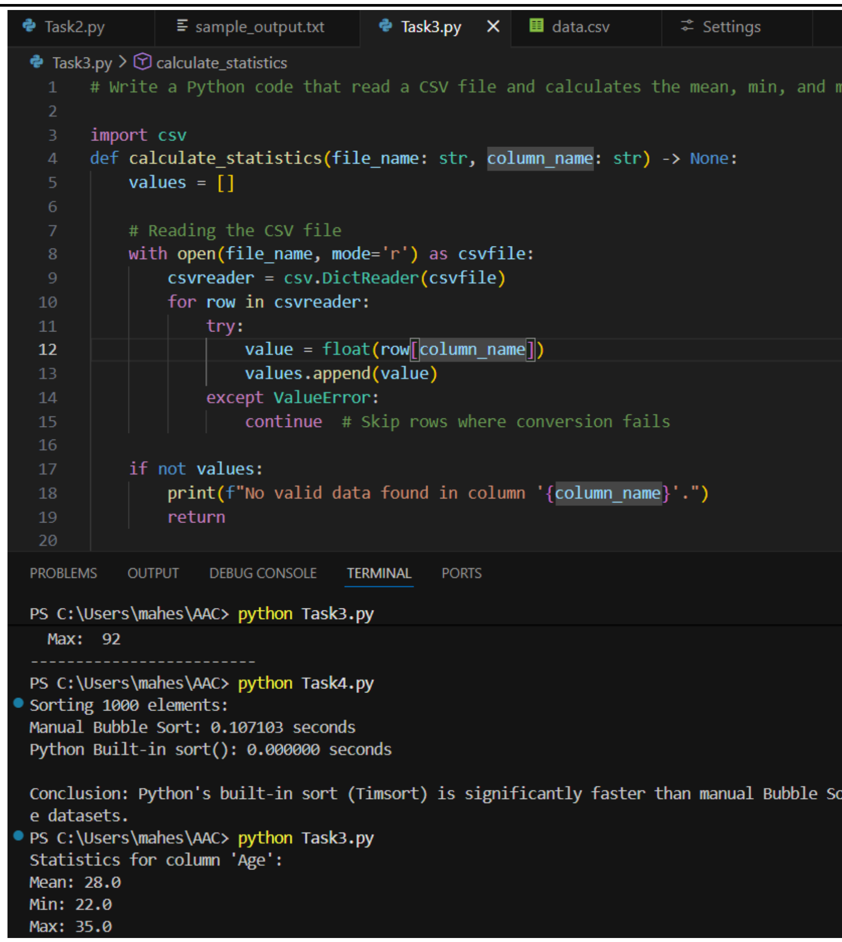
```python
    print(f"Statistics for column
'{column_name}':")
    print(f"Mean: {mean_value}")
    print(f"Min: {min_value}")
    print(f"Max: {max_value}")
if __name__ == "__main__":
    # Example usage - Using 'Age' column from
data.csv
    calculate_statistics('data.csv', 'Age')
```

- Correct output



```
PS C:\Users\mahes\AAC> python -c "import pandas; print(pandas.__ver
2.3.3
PS C:\Users\mahes\AAC> python Task3.py
--- CSV Data Statistics ---
Column: Age
  Mean: 28.00
  Min:  22
  Max:  35
-------------------------
Column: Score
  Mean: 86.60
  Min:  78
  Max:  92
-------------------------
```

- Screenshot



Task2.py | sample_output.txt | Task3.py | data.csv | Settings

Task3.py > calculate_statistics

```python
# Write a Python code that read a CSV file and calculates the mean, min, and ma

import csv
def calculate_statistics(file_name: str, column_name: str) -> None:
    values = []

    # Reading the CSV file
    with open(file_name, mode='r') as csvfile:
        csvreader = csv.DictReader(csvfile)
        for row in csvreader:
            try:
                value = float(row[column_name])
                values.append(value)
            except ValueError:
                continue  # Skip rows where conversion fails

    if not values:
        print(f"No valid data found in column '{column_name}'.")
        return

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\mahes\AAC> python Task3.py
  Max:  92
-------------------------
PS C:\Users\mahes\AAC> python Task4.py
Sorting 1000 elements:
Manual Bubble Sort: 0.107103 seconds
Python Built-in sort(): 0.000000 seconds

Conclusion: Python's built-in sort (Timsort) is significantly faster than manual Bubble Sor
e datasets.
PS C:\Users\mahes\AAC> python Task3.py
Statistics for column 'Age':
Mean: 28.0
Min: 22.0
Max: 35.0
```

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:**

  - Prompt

```
Write a Python code that implements the Bubble
Sort algorithm and compares its execution time
with built in sort method using a list of 5000
random integers.
```

  - Two versions of code

```
import time
import random

def bubble_sort(arr):
    n = len(arr)
    # Manual Bubble Sort Implementation
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1],
arr[j]
    return arr

def compare_sorting_methods():
    # Generate a random list of integers
    size = 1000
    original_data = [random.randint(0, 5000) for _
in range(size)]

    # 1. Bubble Sort (Manual)
    data_bubble = original_data.copy()
    start_time = time.time()
    bubble_sort(data_bubble)
    bubble_duration = time.time() - start_time

    # 2. Built-in sort() (Timsort)
    data_builtin = original_data.copy()
    start_time = time.time()
    data_builtin.sort()
    builtin_duration = time.time() - start_time
```

```python
    print(f"Sorting {size} elements:")
    print(f"Manual Bubble Sort: {bubble_duration:.6f} seconds")
    print(f"Python Built-in sort(): {builtin_duration:.6f} seconds")
    print(f"\nConclusion: Python's built-in sort (Timsort) is significantly "
          f"faster than manual Bubble Sort for large datasets.")

if __name__ == "__main__":
    compare_sorting_methods()
```

- Short comparison

```
Sorting 1000 elements:
Manual Bubble Sort: 0.126919 seconds
Python Built-in sort(): 0.000098 seconds
Conclusion: Python's built-in sort (Timsort) is significantly faster than manual Bubble Sort for large datasets.
```

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.