SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M. Tech/MCA		**Assignment Type: Lab**	**AcademicYear:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code		**Course Title**	AI Assisted Problem Solving Using Python
Year/Sem	I/I	**Regulation**	R24
Date and Day of Assignment	Week1 - TUESDAY	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	M. Tech/MCA
AssignmentNumber:2.3(Present assignment number)/**24**(Total number of assignments)			

Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI

Task Description#1

- Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.

Expected Output#1

- Functional code with output and screenshot

PROMPT-1: Write a python function that reads a CSV file and calculates mean, min, max.

CODE:

```
from typing import Optional, List, Dict
import csv
import statistics
import sys

def csv_numeric_stats(path: str,
                      columns: Optional[List[str]] = None,
                      delimiter: str = ',',
                      header: bool = True) -> Dict[str, Dict[str, Optional[float]]]:
    """
    Read CSV at `path` and compute mean, min, max for numeric values.
    - columns: list of column names to compute. If None, compute for all numeric columns.
    - header: True if first row is header (use names). If False and columns provided, columns should be indices as strings.
    Returns a dict: { column_name_or_index: {"mean":..., "min":..., "max":..., "count":...} }
    Non-numeric or empty cells are ignored.
    """
    data = {}  # column -> list of floats
    with open(path, newline='', encoding='utf-8') as f:
        reader = csv.reader(f, delimiter=delimiter)
        header_row = None
        if header:
            try:
                header_row = next(reader)
            except StopIteration:
                return {}

        # Map requested column names to indices
        col_indices = None
        if columns is not None:
            if header:
                name_to_idx = {name: idx for idx, name in enumerate(header_row)}
                col_indices = []
                for c in columns:
```

```
                    for c in columns:
                        if c in name_to_idx:
                            col_indices.append((c, name_to_idx[c]))
                        else:
                            raise ValueError(f"Column name not found in header: {c}")
                else:
                    # when no header, user should pass indices as strings or ints
                    col_indices = []
                    for c in columns:
                        idx = int(c)
                        col_indices.append((str(idx), idx))
            else:
                # collect all columns (use header names if present, else indices)
                # Peek first data row to know number of columns
                try:
                    first = next(reader)
                except StopIteration:
                    return {
                ncols = len(first)
                # initialize data with header names or index strings
                if header:
                    for i, name in enumerate(header_row):
                        data[name] = []
                else:
                    for i in range(ncols):
                        data[str(i)] = []
                # process the first row
                row = first
                for i, val in enumerate(row):
                    key = header_row[i] if header else str(i)
                    try:
                        v = float(val)
```

(variable) first: list[str]

```
                    v = float(val)
                    data[key].append(v)
                except Exception:
                    pass
            # continue with the rest
            for row in reader:
                for i, val in enumerate(row):
                    key = header_row[i] if header else str(i)
                    try:
                        v = float(val)
                        data[key].append(v)
                    except Exception:
                        pass
            # compute and return
            results = {}
            for key, vals in data.items():
                if vals:
                    results[key] = {
                        "mean": statistics.mean(vals),
                        "min": min(vals),
                        "max": max(vals),
                        "count": len(vals)
                    }
                else:
                    results[key] = {"mean": None, "min": None, "max": None, "count": 0}
            return results

        # If columns were specified, initialize data for them
        for name, idx in col_indices:
            data[name] = []
```

```
        for row in reader:
            for name, idx in col_indices:
                if idx < len(row):
                    val = row[idx].strip()
                    if val == '':
                        continue
                    try:
                        v = float(val)
                        data[name].append(v)
                    except Exception:
                        # ignore non-numeric
                        pass
    compute stats
    results = {}
    for key, vals in data.items():
        if vals:
            results[key] = {
                "mean": statistics.mean(vals),
                "min": min(vals),
                "max": max(vals),
                "count": len(vals)
            }
        else:
            results[key] = {"mean": None, "min": None, "max": None, "count": 0}
    return results

__name__ == "__main__":
    # default CSV path (used if no CLI arg provided)
    path = r"c:\Users\chhar\Downloads\MTECH STUDIES\AIPP ASSIGNMENTs\Assignment-2\sample.csv"
    cols = None
```

```python
# override defaults if arguments provided
if len(sys.argv) >= 2:
    path = sys.argv[1]
if len(sys.argv) >= 3:
    cols = sys.argv[2].split(',')

stats = csv_numeric_stats(path, columns=cols, delimiter=',', header=True)
for col, s in stats.items():
    print(f"{col}: mean={s['mean']}, min={s['min']}, max={s['max']}, count={s['count']}")
```

```
Users/chhar/Downloads/MTECH STUDIES/AIPP ASSIGNMENTs/Assignment-2/Task-1.py"
name: mean=None, min=None, max=None, count=0
age: mean=30.5, min=25.0, max=40.0, count=4
score: mean=89.5, min=85.0, max=95.0, count=4
```

Task Description#2

- Compare Gemini and Copilot outputs for a palindrome check function.

Expected Output#2

- Side-by-side comparison and observations

PROMPT-1: write a python function for a palindrome check function.

Copilot Code:

```python
def is_palindrome(text):
    text = ''.join(e.lower() for e in text if e.isalnum())
    return text == text[::-1]

# Take user input
string = input("Enter a string: ")
if is_palindrome(string):
    print("It is a palindrome")
else:
    print("Not a palindrome")
```

Explanation:

- Converts all characters to lowercase.
- Removes all non-alphanumeric characters (punctuation, spaces, etc.).
- Compares the cleaned string with its reverse.
- Returns True if palindrome, otherwise False

Gemini Code:

```python
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]

# Take user input
text = input("Enter a string: ")
if is_palindrome(text):
    print("It is a palindrome")
else:
    print("Not a palindrome")
```

Explanation:

- Converts the string to lowercase.
- Removes spaces.
- Compares the string with its reverse (s[::-1]).

- Returns True if both are equal; otherwise, False.

Observations

- Gemini's version is simpler and best suited for basic palindrome checks (like "madam" or "racecar").
- Copilot's version is more robust, handling cases like punctuation and numbers (e.g., "A man, a plan, a canal: Panama").
- Both produce correct results for simple inputs, but Copilot's approach is more production-ready due to better input sanitization.

Side-by-Side Comparison

Feature / Aspect	Gemini Output	**Copilot Output**
Variable Name	s	text
Cleaning Method	Removes spaces only	Removes all non-alphanumeric characters
Case Handling	Converts to lowercase	Converts to lowercase
Reversal Method	s[::-1]	text[::-1]
Accuracy	sentences without punctuation	Works for sentences with punctuation and numbers
Code Length	Shorter and simpler	Slightly longer but more robust
Readability	Easy to understand	Slightly complex but cleaner input handling

Task Description#3

- Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..

Expected Output#3

- Detailed explanation with code snippet

PROMPT-1: explain a Python function (to calculate area of various shapes) line by line.Detailed explanation with code snippet

PROMPT-2: Take Input from User

```
k3.py > ...
# ...existing code...
import math
from typing import Literal

def area_circle(radius: float) -> float:
    """Return area of a circle given its radius."""
    if radius < 0:
        raise ValueError("radius must be non-negative")
    return math.pi * radius * radius

def area_rectangle(width: float, height: float) -> float:
    """Return area of a rectangle given width and height."""
    if width < 0 or height < 0:
        raise ValueError("width and height must be non-negative")
    return width * height

def area_triangle(base: float, height: float) -> float:
    """Return area of a triangle given base and height."""
    if base < 0 or height < 0:
        raise ValueError (parameter) height: float on-negative")
    return 0.5 * base * height

def area_trapezoid(a: float, b: float, height: float) -> float:
    """Return area of a trapezoid with parallel sides a and b and given height."""
    if a < 0 or b < 0 or height < 0:
        raise ValueError("sides and height must be non-negative")
    return 0.5 * (a + b) * height
```

```
    return 0.5 * base * height

def area_trapezoid(a: float, b: float, height: float) -> float:
    """Return area of a trapezoid with parallel sides a and b and given height."""
    if a < 0 or b < 0 or height < 0:
        raise ValueError("sides and height must be non-negative")
    return 0.5 * (a + b) * height

def area_ellipse(a: float, b: float) -> float:
    """Return area of an ellipse with semi-axes a and b."""
    if a < 0 or b < 0:
        raise ValueError("semi-axes must be non-negative")
    return math.pi * a * b

def area(shape: Literal['circle','rectangle','triangle','trapezoid','ellipse'], **kwargs)
    """
    Generic dispatcher to compute area for supported shapes.
    Call examples:
      area('circle', radius=3)
      area('rectangle', width=4, height=5)
    """
    if shape == 'circle':
        return area_circle(kwargs['radius'])
    if shape == 'rectangle':
        return area_rectangle(kwargs['width'], kwargs['height'])
    if shape == 'triangle':
        return area_triangle(kwargs['base'], kwargs['height'])
    if shape == 'trapezoid':
```

```
sers\chhar\Downloads\MTECH STUDIES\AIPP ASSIGNMENTs\Assignment-2\Task3.py
    print("Supported shapes: circle, rectangle, triangle, trapezoid, ellipse")
    shape = input("Enter shape: ").strip().lower()
    try:
        if shape == 'circle':
            r = float(input("Enter radius: ").strip())
            print(f"Area = {area('circle', radius=r)}")
        elif shape == 'rectangle':
            w = float(input("Enter width: ").strip())
            h = float(input("Enter height: ").strip())
            print(f"Area = {area('rectangle', width=w, height=h)}")
        elif shape == 'triangle':
            b = float(input("Enter base: ").strip())
            h = float(input("Enter height: ").strip())
            print(f"Area = {area('triangle', base=b, height=h)}")
        elif shape == 'trapezoid':
            a = float(input("Enter side a: ").strip())
            b = float(input("Enter side b: ").strip())
            h = float(input("Enter height: ").strip())
            print(f"Area = {area('trapezoid', a=a, b=b, height=h)}")
        elif shape == 'ellipse':
            a = float(input("Enter semi-axis a: ").strip())
            b = float(input("Enter semi-axis b: ").strip())
            print(f"Area = {area('ellipse', a=a, b=b)}")
        else:
            print("Unsupported shape")
    except (ValueError, KeyError) as e:
        print("Error:", e)
...existing code...
```

Explaination:

- import math
  - Pulls in the math module to use math.pi for accurate π and other math utilities.
- from typing import Literal
  - Imports Literal for the shape parameter type hint in the dispatcher (constrains allowed shape strings).
- def area_circle(radius: float) -> float:
  - Declares function name, a typed parameter radius, and a typed return value float.
- """Return area of a circle given its radius."""
  - Docstring briefly describing the function.
- if radius < 0:
  raise ValueError("radius must be non-negative")
  - Validates input and raises a clear error on invalid (negative) radius.
- return math.pi * radius * radius
  - Calculates and returns area using formula π * r^2.
- def area_rectangle(width: float, height: float) -> float:
  - Function to compute rectangle area with width and height parameters.
- if width < 0 or height < 0:
  raise ValueError("width and height must be non-negative")
  - Input validation for non-negative dimensions.
- return width * height
  - Computes area as width × height.
- def area_triangle(base: float, height: float) -> float:

    - Function to compute triangle area.
- if base < 0 or height < 0:
  raise ValueError("base and height must be non-negative")
    - Validate inputs.
- return 0.5 * base * height
    - Computes triangle area 1/2 × base × height.
- def area_trapezoid(a: float, b: float, height: float) -> float:
    - Function to compute trapezoid area with parallel sides a and b.
- if a < 0 or b < 0 or height < 0:
  raise ValueError("sides and height must be non-negative")
    - Validate inputs.
- return 0.5 * (a + b) * height
    - Area formula: average of parallel sides times height.
- def area_ellipse(a: float, b: float) -> float:
    - Function for ellipse area using semi-axes a and b.
- if a < 0 or b < 0:
  raise ValueError("semi-axes must be non-negative")
    - Validate inputs.
- return math.pi * a * b
    - Ellipse area formula π × a × b.
- def area(shape: Literal[...], **kwargs) -> float:
    - Generic dispatcher function. shape selects shape; kwargs holds required numeric arguments.
- The series of if shape == ... lines
    - Each branch extracts expected keyword arguments from kwargs and calls the specific shape function.
    - If a required kwarg is missing, Python raises KeyError — making missing inputs explicit.
    - This keeps the dispatcher small and delegates validation to the specific functions.
- raise ValueError(f"Unsupported shape: {shape}")
    - If the provided shape string is not supported, a clear error is raised

Task Description#4

- Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).

Expected Output#4

- Screenshots of working environments with few prompts to generate python code

PROMPT: Generate a python code to find sum of squares and take input from user

```python
# ...existing code...
from typing import Iterable

def sum_of_squares(nums: Iterable[float]) -> float:
    """
    Return the sum of squares of numeric values in nums.
    Non-numeric values are ignored. Accepts any iterable of values.
    """
    total = 0.0
    for x in nums:
        try:
            n = float(x)
        except (TypeError, ValueError):
            continue
        total += n * n
    return total

if __name__ == "__main__":
    raw = input("Enter numbers (comma or space separated), or press Enter to use sample
    if raw:
        parts = raw.replace(',', ' ').split()
        try:
            nums = [float(p) for p in parts]
        except ValueError:
            print("Invalid input — please enter numbers only.")
            raise SystemExit(1)
    else:
        nums = [1, 2, 3, 4]

    print(f"Numbers: {nums}")
    print("Sum of squares:", sum_of_squares(nums))
# ...existing code...
```

OUTPUT:

```
ASSIGNMENTs/Assignment-2/task-4.py"
Enter numbers (comma or space separated), or press Enter to use sample [1,2,3,4]:
Numbers: [1, 2, 3, 4]
Sum of squares: 30.0
```

Task Description#5

- Student need to write code to calculate sum of add number and even numbers in the list

Expected Output#5

- Refactored code written by student with improved logic

PROMPT-1: generate a code to calculate the sum of odd numbers and even numbers in the list and take input from user

```python
import re
from typing import Iterable, List, Dict

def parse_ints_from_string(s: str) -> List[int]:
    """
    Parse integers from a string containing numbers separated by commas/spaces.
    Non-integer tokens are ignored.
    """
    if not s:
        return []
    parts = re.split(r'[,\s]+', s.strip())
    ints: List[int] = []
    for p in parts:
        if p == '':
            continue
        try:
            ints.append(int(p))
        except ValueError:
            # ignore tokens that are not integers
            continue
    return ints

def sum_even_odd(nums: Iterable[int]) -> Dict[str, int]:
    """
    Compute sums of even and odd integers in a single pass.
    Returns {"even": sum_of_evens, "odd": sum_of_odds}.
    """
    even_sum = 0
    odd_sum = 0
    for n in nums:
        try:
            i = int(n)
        except (TypeError, ValueError):
```

Ln 10, Col 14 Sp

```python
        try:
            i = int(n)
        except (TypeError, ValueError):
            continue
        if i % 2 == 0:
            even_sum += i
        else:
            odd_sum += i
    return {"even": even_sum, "odd": odd_sum}

if __name__ == "__main__":
    raw = input("Enter integers (comma or space separated), or press Enter to use [1 2 3 4
    if raw:
        numbers = parse_ints_from_string(raw)
        if not numbers:
            print("No valid integers parsed. Exiting.")
        else:
            res = sum_even_odd(numbers)
            print(f"Numbers: {numbers}")
            print(f"Sum of even numbers: {res['even']}")
            print(f"Sum of odd numbers:  {res['odd']}")
    else:
        sample = [1, 2, 3, 4, 5, 6]
        res = sum_even_odd(sample)
        print(f"Sample Numbers: {sample}")
        print(f"Sum of even numbers: {res['even']}")
        print(f"Sum of odd numbers:  {res['odd']}")
...existing code...
```

```
ASSIGNMENTs/Assignment-2/Task-5.py
Enter integers (comma or space separated), or press Enter to use [1 2 3 4 5 6]:
Sample Numbers: [1, 2, 3, 4, 5, 6]
Sum of even numbers: 12
Sum of odd numbers:  9
PS C:\Users\chhar\Downloads\MTECH STUDIES\AIPP ASSIGNMENTs\Assignment-2>
```

Evaluation Criteria:

Criteria	Max Marks
Successful Use of Gemini in Colab (Task#1 & #2)	2.5
Code Explanation Accuracy (Gemini) (Task#3)	2.5
Cursor AI Setup and Usage (Task#4)	2.5
Refactoring and Improvement Analysis (Task#5)	2.5
Total	**10 Marks**