SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Course Title – AI Assisted Coding

Course Code : 24CS002PC215

Assignment Type : Lab

Academic Year : 2025–2026

Name : Soumith Sripathi

Enroll Number : 2403A51340

Batch Number : 24BTCAICSB14

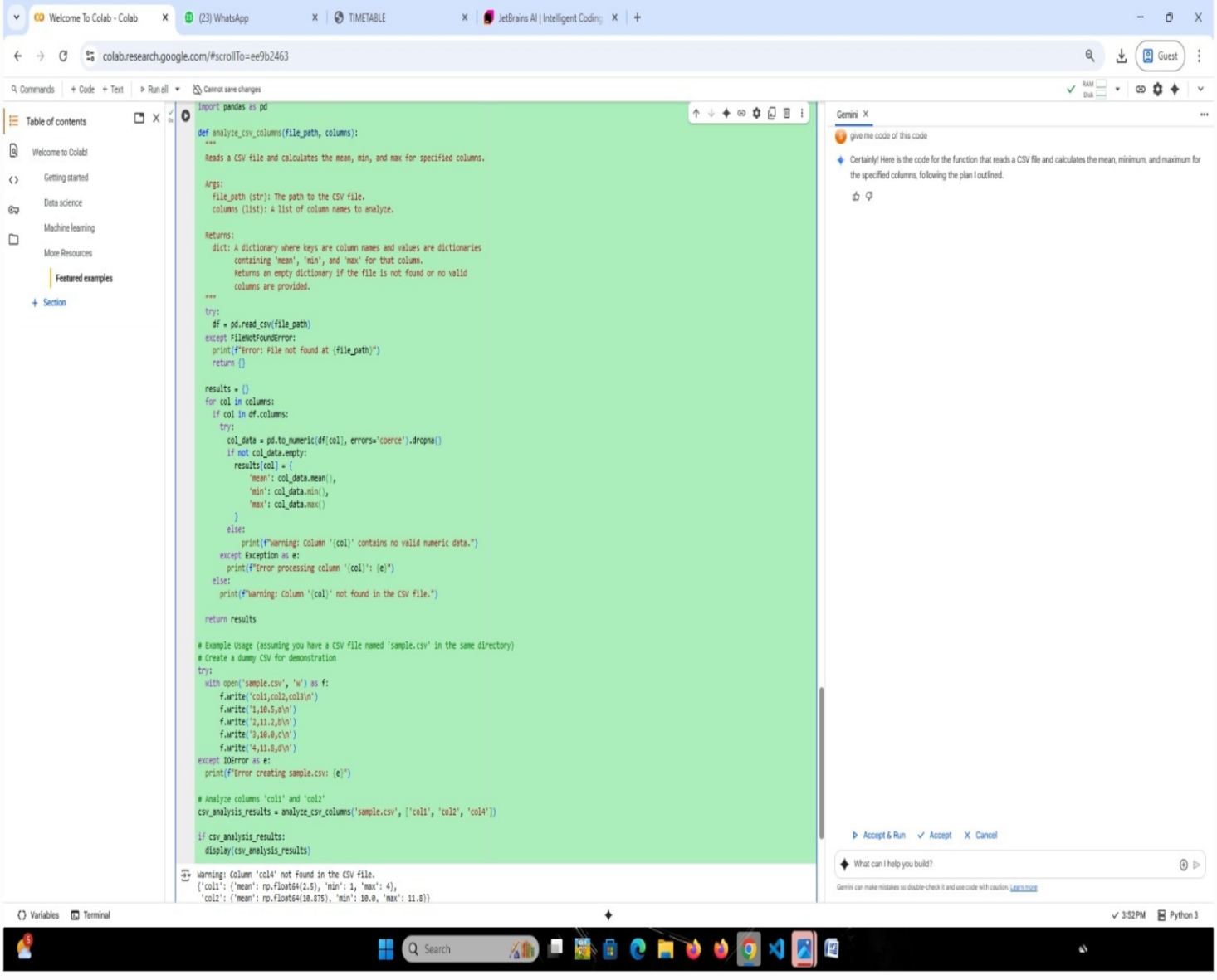
Week – 2

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

Task Description – 1 :

Question–1 : Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.

CODE & OUTPUT :



Explanation Of The Above Program :

~ Loads the pandas library (short name pd) which is used to read CSVs and compute statistics.

~ Reads a CSV file and calculates the mean, min, and max for specified columns.

~ Tries to load the CSV into a DataFrame df.

~ If the file path is wrong, it prints an error and returns an empty dict so the caller knows nothing could be computed.

~ This makes the function robust to mixed–type columns (strings, empty cells, etc.)

~ If anything else goes wrong while processing a column (rare), it prints an error message including the exception.

~ The function returns stats for the columns that do exist and contain numeric data.

~ Sample CSV rows created by the script (header + 4 rows):
col1 values: 1, 2, 3, 4
Sum = 1 + 2 + 3 + 4 = 10
Mean = 10 / 4 = 2.5
Min = 1, Max = 4

Task Description – 2 :

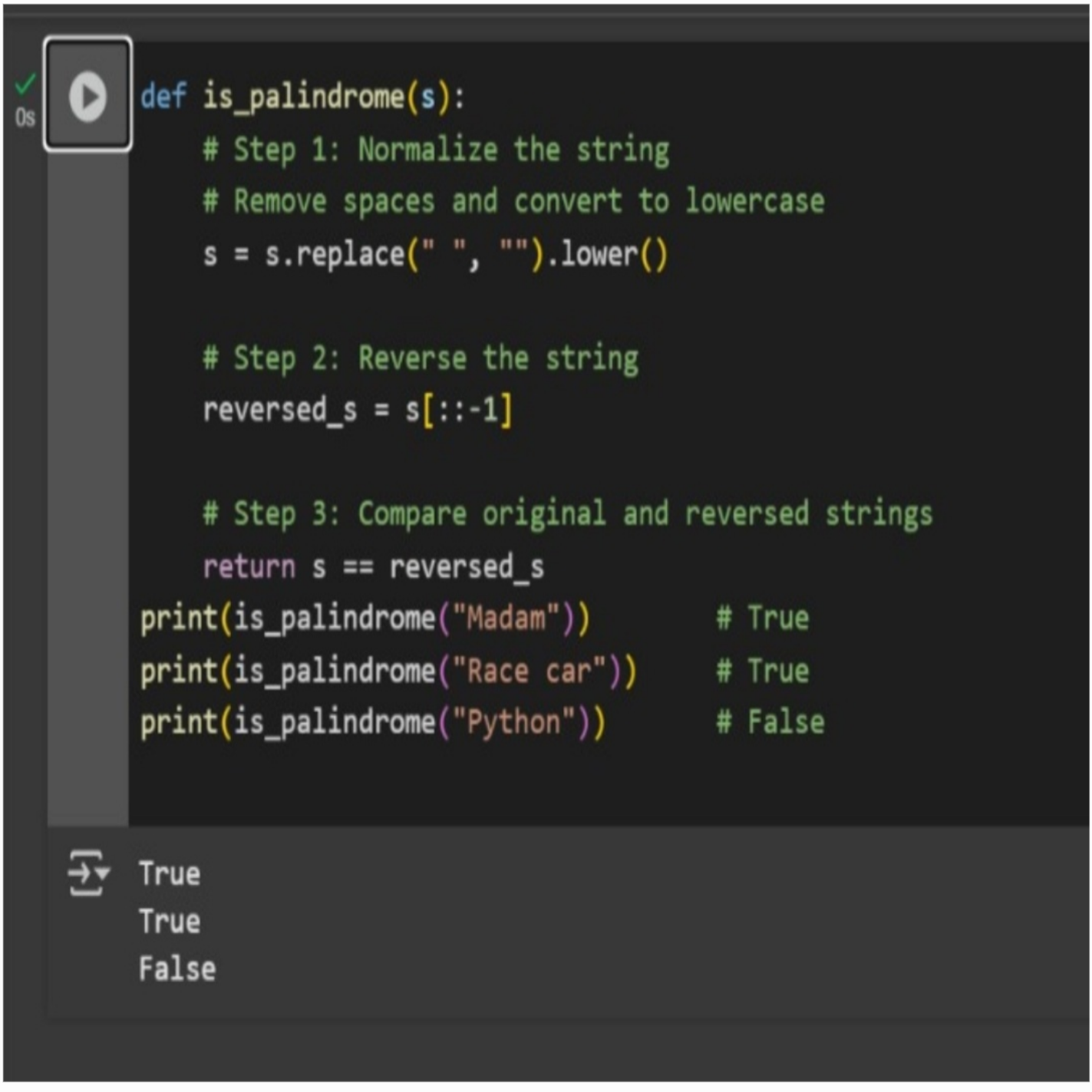
Question–2 : Compare Gemini and Copilot outputs for a palindrome check function.

CODE & OUTPUT : (Gemini Ai)

```
[9]  def is_palindrome_number(number):
       """
       Checks if a number is a palindrome.

       A palindrome is a number that reads the same backward as forward.

       Args:
         number: An integer.

       Returns:
         bool: True if the number is a palindrome, False otherwise.
       """
       # Convert the number to a string
       num_str = str(number)
       # Check if the string is equal to its reverse
       return num_str == num_str[::-1]

     # Example usage:
     print(f"Is 121 a palindrome? {is_palindrome_number(121)}")
     print(f"Is 123 a palindrome? {is_palindrome_number(123)}")
     print(f"Is 1221 a palindrome? {is_palindrome_number(1221)}")
     print(f"Is -101 a palindrome? {is_palindrome_number(-101)}")
```

```
Is 121 a palindrome? True
Is 123 a palindrome? False
Is 1221 a palindrome? True
Is -101 a palindrome? False
```

CODE & OUTPUT :(Copilot)



```python
def is_palindrome(s):
    # Step 1: Normalize the string
    # Remove spaces and convert to lowercase
    s = s.replace(" ", "").lower()

    # Step 2: Reverse the string
    reversed_s = s[::-1]

    # Step 3: Compare original and reversed strings
    return s == reversed_s
print(is_palindrome("Madam"))        # True
print(is_palindrome("Race car"))     # True
print(is_palindrome("Python"))       # False
```

```
True
True
False
```

Comparison of both outputs :

Aspect	First Program (Number Palindrome)	Second Program (String Palindrome)
Function Name	`is_palindrome_number`	`is_palindrome`
Input Parameter	`number` (integer)	`s` (string)
Purpose	Checks if a number reads the same forward and backward.	Checks if a string reads the same forward and backward, ignoring spaces and letter case.
Preprocessing	Converts the integer to a string with `str(number)`.	Removes spaces with `.replace(" ", "")` and converts to lowercase with `.lower()`.
Reverse Logic	Uses slicing `[::-1]` to reverse the string form of the number.	Uses slicing `[::-1]` to reverse the normalized string.
Comparison	Compares the string form of the number to its reverse.	Compares the normalized string to its reverse.
Return Value	Boolean: `True` if palindrome, else `False`.	Boolean: `True` if palindrome, else `False`.
Special Handling	Works with integers, but negative numbers are not palindromes because of the minus sign.	Works with phrases, ignores spaces and capitalization, but doesn't ignore punctuation.
Example Inputs & Outputs	`121 → True`, `123 → False`, `1221 → True`, `-101 → False`	`"Madam" → True`, `"Race car" → True`, `"Python" → False`
Key Use Case	Numeric palindrome checking (IDs, codes,	Text palindrome checking (words,

Explanation Of The Above Program :

~ function named is_palindrome_number that takes one parameter number.

~ The triple–quoted block right under the def is the docstring. It describes what the function does, its arguments (number: An integer) and what it returns (bool: True if ...).

~ Converts the integer (or any input) to its string representation and stores it in num_str.

~ Check if the string is equal to its reverse return num_str == num_str[::–1] num_str[::–1] is Python slice syntax that produces the reversed string (start:end:step where step = –1).

~ The == compares the original string to its reversed version and yields True or False. That boolean is returned as the function result.

~ These use f–strings to embed the function result into printed text.

Task Description – 3 :

Question–3 : Ask Gemini to explain a Python function (to calculate area of various shapes) line by line.

CODE & OUTPUT :

```python
import math

def calculate_area(shape, **kwargs):
  """Calculates the area of various shapes.

  Args:
    shape: A string representing the shape ('circle' or 'rectangle').
    **kwargs: Keyword arguments for the dimensions of the shape.
      For a circle, pass 'radius'.
      For a rectangle, pass 'length' and 'width'.

  Returns:
    The area of the shape, or None if the shape is not supported or
    dimensions are missing.
  """
  if shape.lower() == 'circle':
    radius = kwargs.get('radius')
    if radius is not None:
      return math.pi * radius**2
    else:
      print("Error: Radius not provided for circle.")
      return None
  elif shape.lower() == 'rectangle':
    length = kwargs.get('length')
    width = kwargs.get('width')
    if length is not None and width is not None:
      return length * width
    else:
      print("Error: Length and/or width not provided for rectangle.")
      return None
  else:
    print(f"Error: Shape '{shape}' not supported.")
    return None

# Example usage:
circle_area = calculate_area('circle', radius=5)
if circle_area is not None:
  print(f"Area of the circle: {circle_area}")

rectangle_area = calculate_area('rectangle', length=4, width=6)
if rectangle_area is not None:
  print(f"Area of the rectangle: {rectangle_area}")

invalid_shape_area = calculate_area('triangle', base=10, height=5)
missing_arg_area = calculate_area('circle')
```

```
Area of the circle: 78.53981633974483
Area of the rectangle: 24
Error: Shape 'triangle' not supported.
Error: Radius not provided for circle.
```

Explanation Of The Above Program :

~ Loads the math module so we can use math.pi (the value of π).

~ Defines a function named calculate_area.

~ shape is a string that tells the function which shape to compute (e.g., 'circle' or 'rectangle').

~ additional named arguments (like radius=5 or length=4, width=6) into a dictionary.

~ Explains the function purpose, expected args for each shape and what it returns.

~ Makes the shape check case–insensitive by converting shape to lowercase.

~ If shape is "circle", the code inside this block runs.

~ Tries to get the radius value from the keyword arguments. get() returns None if radius wasn't provided.

~ If radius was provided, compute the circle area using and return it.

~ for radius=5, returned value is math.pi * 25 → 78.53981633974483.

~ If no radius was provided, print an error message "Error: Radius not provided for circle." and return None.

~ If shape is "rectangle", handle rectangle area next.

~ Retrieve length and width from kwargs (or None if missing).

~ If both dimensions are present, compute and return area = length × width (e.g., 4 * 6 = 24).

~ If either length or width is missing, print "Error: Length and/or width not provided for rectangle." and return None.

~If shape is not 'circle' or 'rectangle', print an error like Error: Shape 'triangle' not supported. and return

None.

~circle_area = calculate_area('circle', radius=5) → stores 78.5398.... The following if circle_area is not None: prints "Area of the circle: 78.5398...".

~rectangle_area = calculate_area('rectangle', length=4, width=6) → prints "Area of the rectangle: 24".

~ invalid_shape_area = calculate_area('triangle', base=10, height=5) → prints "Error: Shape 'triangle' not supported." (function returned None).

~ missing_arg_area = calculate_area('circle') → prints "Error: Radius not provided for circle." (function returned None).

~ Program output (as shown):
Area of the circle: 78.53981633974483
Area of the rectangle: 24
Error: Shape 'triangle' not supported.
Error: Radius not provided for circle.

Task Description – 4 :

Question–4 : Use it to generate a Python function (e.g., sum of squares)

CODE & OUTPUT :

```
[7] def sum_of_squares(numbers):
      """
      Calculates the sum of squares of a list of numbers.

      Args:
        numbers (list): A list of numbers.

      Returns:
        int or float: The sum of the squares of the numbers.
      """
      total_sum = 0
      for number in numbers:
        total_sum += number**2
      return total_sum

    # Example usage:
    my_list = [1, 2, 3, 4, 5]
    result = sum_of_squares(my_list)
    print(f"The sum of squares of {my_list} is: {result}")

    my_list_2 = [1.5, 2.5, 3.5]
    result_2 = sum_of_squares(my_list_2)
    print(f"The sum of squares of {my_list_2} is: {result_2}")
```

```
The sum of squares of [1, 2, 3, 4, 5] is: 55
The sum of squares of [1.5, 2.5, 3.5] is: 20.75
```

Explanation Of The Above Program :

~ sum_of_squares(numbers) that computes and returns the sum of each item in numbers squared, then demonstrates the function with two example lists.

~ def sum_of_squares(numbers): this creates a function named sum_of_squares that expects one argument numbers (a list or iterable of numeric values).

~ The triple–quoted string directly under the def explains the function's purpose, the expected arguments (numbers is a list of numbers), and what it returns (an int or float that is the sum of squares).

~ total_sum = 0 — starts an accumulator variable at 0. This will hold the running total of squares.

~ for number in numbers: — iterates through each element in the input list one by one.

~total_sum += number**2 — squares the current

number using **2, then adds that squared value to total_sum. The += is shorthand for total_sum = total_sum + (number**2).

~ return total_sum — after the loop finishes, the function returns the accumulated sum of squares.

~ For the integer function computes $1^2+2^2+3^2+4^2+5^2 = 55$ and prints it.

~For the floats function computes $1.5^2 + 2.5^2 + 3.5^2 = 2.25 + 6.25 + 12.25 = 20.75$ and prints it.

Task Description – 5 :

Question–5 : write code to calculate sum of odd numbers and even numbers in the list

CODE & OUTPUT :

un all ▾ Cannot save changes

[8]
```python
def sum_odd_even(numbers):
  """
  Calculates the sum of odd and even numbers in a list.

  Args:
    numbers (list): A list of numbers.

  Returns:
    tuple: A tuple containing the sum of odd numbers and the sum of even numbers.
  """
  sum_odd = 0
  sum_even = 0
  for number in numbers:
    if number % 2 == 0:
      sum_even += number
    else:
      sum_odd += number
  return sum_odd, sum_even

# Example usage:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_sum, even_sum = sum_odd_even(my_list)

print(f"Original list: {my_list}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")

my_list_2 = [15, 22, 33, 48, 57]
odd_sum_2, even_sum_2 = sum_odd_even(my_list_2)

print(f"\nOriginal list: {my_list_2}")
print(f"Sum of odd numbers: {odd_sum_2}")
print(f"Sum of even numbers: {even_sum_2}")
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30

Original list: [15, 22, 33, 48, 57]
Sum of odd numbers: 105
Sum of even numbers: 70
```

Explanation Of The Above Program :

~ sum_odd_even(numbers) that separates numbers into odd and even, computes the sum of each group, and returns both sums as a tuple

(sum_odd, sum_even).

~ def sum_odd_even(numbers): —declares the function and expects one argument numbers (a list or iterable of numeric values).

~ The triple–quoted string under the header explains the purpose, argument (numbers), and return type (a tuple with odd–sum and even–sum).

~ Initialize accumulators
sum_odd = 0 ;sum_even = 0
Start both sums at 0. They will hold the running totals.

~ The function examines each value in numbers one by one.

~ tests whether number is divisible by 2 (even). If even, add it to sum_even; otherwise add it to sum_odd.

~ my_list = [1,2,3,4,5,6,7,8,9,10]

Odd numbers: 1,3,5,7,9 → 1+3=4; 4+5=9; 9+7=16; 16+9=25 → sum_odd = 25.

Even numbers: 2,4,6,8,10 → 2+4=6; 6+6=12; 12+8=20; 20+10=30 → sum_even = 30

Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of odd numbers: 25
Sum of even numbers: 30

~ my_list_2 = [15, 22, 33, 48, 57]

Odd numbers: 15+33+57 → 15+33=48; 48+57=105 → sum_odd = 105.

Even numbers: 22+48 → sum_even = 70.

~ Original list: [15, 22, 33, 48, 57]
Sum of odd numbers: 105
Sum of even numbers: 70.