AI ASSISTED CODING

Name: ORSU VENNELA

Ht.No:2303A52418

Batch:35

Assignment2.1

Task-1:

Task1:StatisticalSummaryforSurveyData

❖ **Scenario:**

Youareadataanalystinternworkingwithsurveyresponses storedas

numericallists.

❖ **Task:**

UseGoogleGeminiinColabtogenerateaPythonfunctionthat reads a list

ofnumbersandcalculatesthemean,minimum,andmaximum values.

Prompt:

GenerateaPythonfunctionthattakesalistofnumbersasinput andcalculatesthemean,minimum,andmaximumvalues.Also show sample output.

Code:

```python
def calculate_stats(numbers):
    """
    Calculates the mean, minimum, and maximum of a list of numbers.
    """
    if not numbers:
        return None, None, None

    min_val = min(numbers)
    max_val = max(numbers)
    mean_val = sum(numbers) / len(numbers)

    return mean_val, min_val, max_val

# Example usage:
data = [10, 20, 30, 40, 50]
mean, minimum, maximum = calculate_stats(data)

print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")
```

Output:

```
Mean: 30.0
Minimum: 10
Maximum: 50
```

Explanation:

This code summarizes a list of numbers by identifying its boundaries and its center.

- Boundary Detection: The `min()` and `max()` functions scan the entire list to find the smallest and largest values.
- Mean Calculation: The code uses `sum(numbers) / len(numbers)` to divide the total of all values by the count of items in the list.
- Crash Prevention: The `if not numbers` check ensures the code returns `None` instead of crashing if an empty list is provided.
- Result Packing: It returns all three values at once, which are then "unpacked" into the variables `mean`, `minimum`, and `maximum` for easy printing.

Task 2: Armstrong Number – AI Comparison

❖ **Scenario:**

You are evaluating AI tools for numeric validation logic.

❖ Task:

Generate an Armstrong number checker using Gemini and GitHub Copilot.

Compare their outputs, logic style, and clarity.

❖ Expected Output:

➢ Side-by-side comparison table

➢ Screenshots of prompts and generated code

Gemini Prompt:

Write a Python program to check whether a number is an Armstrong number.

Code:

```python
def is_armstrong(number):
    # Convert number to string to easily iterate through digits
    digits = str(number)
    num_digits = len(digits)

    # Calculate the sum of digits raised to the power of num_digits
    total_sum = sum(int(digit) ** num_digits for digit in digits)

    return total_sum == number

# Test the function
num = int(input("Enter a number: "))
if is_armstrong(num):
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

Output:

```
...    Enter a number: 153
       153 is an Armstrong number.
```

GitHub Copilot:

#ArmstrongNumberChecker

Code:

```python
armstrong.py >  armstrong.py
1   # Armstrong number checker
2   def is_armstrong_number(number):
3       # Convert the number to string to easily iterate over digits
4       digits = str(number)
5       num_digits = len(digits)
6
7       # Calculate the sum of each digit raised to the power of num_digits
8       sum_of_powers = sum(int(digit) ** num_digits for digit in digits)
9
10      # An Armstrong number is one where the sum of the powers equals the original number
11      return sum_of_powers == number
12  # Example usage
13  number = 153
14  if is_armstrong_number(number):
15      print(f"{number} is an Armstrong number.")
16  else:
17      print(f"{number} is not an Armstrong number.")
18  # Output:
19  # 153 is an Armstrong number.
```

Output:

Example usage

153 is an Armstrong number.

Comparisontable:

AI Coding Tools: A Comparison

Feature	G Google Gemini AI		C Google Colab
Purpose	AI assistant for code generation, and problem solving	Cloud-based platform write xecutio environment	Cloud-based platform to and execute Python code
Code Generation	Generates code using natural language promts	Does not generate byself	Does not generate code byself
Code Explanation	Explains logic fof step promts	No-builh explanation feature	No-builln explanation feature
Userution of Code	Prompt-based (chat-style interaction)	Cannot execute code	Notebook-baed (cells for code and output)
Error Detection	Suggests fixes and improvements	Medium – requires coding knowlege	Shows runtime and syntax errors
Best Use Case Internet & Libraries	Understanding logic, generating code, learning learning concepts	Running programs, testing outputs, data analysis	Supports Python libraries like Nupmu, Pandas, Matèrplotic

Task3:LeapYearValidationUsingCursorAI

❖ **Scenario:**

Youarevalidatingacalendarmoduleforabackendsystem.

❖ **Task:**

UseCursorAItogenerateaPythonprogramthatcheckswhetheragiven year is

a leap year.

Useatleasttwodifferentpromptsandobservechangesin code.

❖ **ExpectedOutput:**

➢ Twoversionsofcode

➢ Sampleinputs/outputs

➢ Brief comparison

Prompt 1:

WriteaPythonprogramtocheckwhetheragivenyearisaleapyear.

Code:

```python
def is_leap_year(year: int) -> bool:
    # Leap year if divisible by 4, except years divisible by 100,
    # unless also divisible by 400.
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

if __name__ == "__main__":
    try:
        year = int(input("Enter a year: ").strip())
        print(f"{year} is a leap year." if is_leap_year(year) else f"{year} is r
    except ValueError:
        print("Please enter a valid integer year.")
```

Output:

```
...   Enter a year: 2025
      2025 is not a leap year.
```

Prompt 2:

Generateawell-commentedPythonprogramtovalidateleapyearwithinput and output examples.

Code:

```python
def is_leap_year(year: int) -> bool:
    """
    Return True if the given Gregorian calendar year is a leap year.

    Rules:
    - Leap if divisible by 4,
    - except years divisible by 100,
    - unless divisible by 400.
    """
    # Divisible by 400 => leap (covers years like 2000)
    if year % 400 == 0:
        return True
    # Divisible by 100 (but not 400) => common year (covers years like 1900)
    if year % 100 == 0:
        return False
    # Divisible by 4 (but not 100) => leap (covers years like 2024)
    if year % 4 == 0:
        return True
    # Otherwise => common year
    return False


def main():
    # Prompt the user; strip to allow clean parsing, then convert to int
    raw = input("Enter a year: ").strip()
    try:
        year = int(raw)
    except ValueError:
        print(f"Invalid year: {raw!r}. Please enter an integer, e.g., 2024.")
        return

    if is_leap_year(year):
        print(f"{year} is a leap year.")
    else:
        print(f"{year} is NOT a leap year.")


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

Output:

```
Enter a year: 2021
2021 is NOT a leap year.
```

📊 Task 3: Comparison Between Two Prompts (Cursor AI – Leap Year Program)

Aspect	Prompt 1	Prompt 2
Prompt Used	*"Write a Python program to check whether a given year is a leap year."*	*"Generate a well-commented Python program to validate leap year with sample input and output."*
Input Method	Takes year as input from the user	Takes year as input from the user
Code Structure	Simple and concise	Structured and well-organized
Use of Comments	Minimal or no comments	Detailed comments explaining each step
Readability	Easy to read	Very easy to understand

Task4:StudentLogic+AIRefactoring(Odd/EvenSum)

❖ Scenario:

CompanypolicyrequiresdeveloperstowritelogicbeforeusingAI.

❖ Task:

WriteaPythonprogramthatcalculatesthesumofoddandevennumbers in a tuple, then refactor it using any AI tool.

❖ ExpectedOutput:

➢ Originalcode

➢ Refactoredcode

➢ Explanationofimprovements

Mycode(Input&Output):

```python
t = (1, 2, 3, 4, 5, 6)
even_sum = 0
odd_sum = 0

for i in t:
    if i % 2 == 0:
        even_sum += i
    else:
        odd_sum += i

print("Even Sum:", even_sum)
print("Odd Sum:", odd_sum)
```

```
Even Sum: 12
Odd Sum: 9
```

RefactoredCode(Input&Output):

```python
numbers_tuple = (1, 2, 3, 4, 5, 6)

even_sum = sum(num for num in numbers_tuple if num % 2 == 0)
odd_sum = sum(num for num in numbers_tuple if num % 2 != 0)

print(f"Refactored Code Output:")
print(f"Even Sum: {even_sum}")
print(f"Odd Sum: {odd_sum}")
```

```
Refactored Code Output:
Even Sum: 12
Odd Sum: 9
```

ImprovementsinAI-RefactoredCode:

1. TherefactoredcodereducesthenumberoflinesbyusingPython'sbuilt- in sum() function.
2. Generatorexpressionsreplacemanualloops,makingthecodemore efficient.
3. Readabilityisimprovedbyusingconciseandexpressivesyntax.
4. Thelogicbecomeseasiertomaintainandmodifyinfuture.
5. Theoutputremainsthesamewhilethecodequalityisenhanced.