SR UNIVERSITY

Program	:B.tech(CSE)
Specialization	:AIML
Course Title	:AI Assisted Coding
Course Code	:24CS002PC215
Semester	:3rd semester
Academic Session	:2025-2026
Name of Student	:Kaveti Manohar
Enrollment No.	:2403A52079
Batch No.	:02
Date	:10/09/2025

LAB ASSIGNMENT

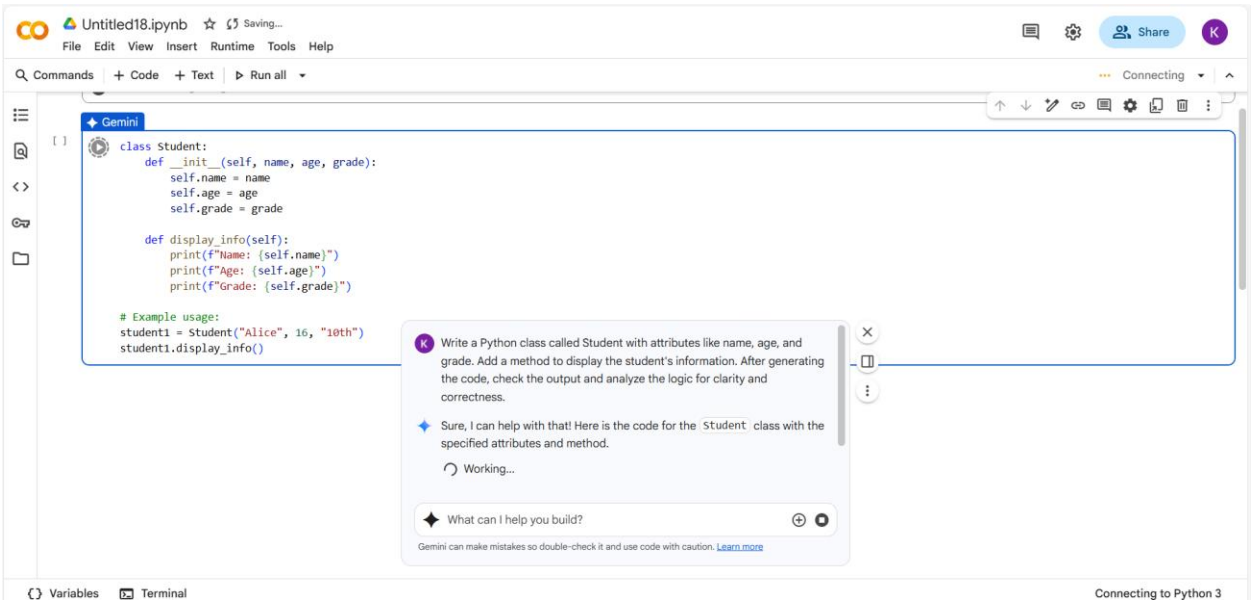
Task Description -1:

Use AI to complete a Student class with attributes and a method.
Check output
Analyze the code generated by AI tool.

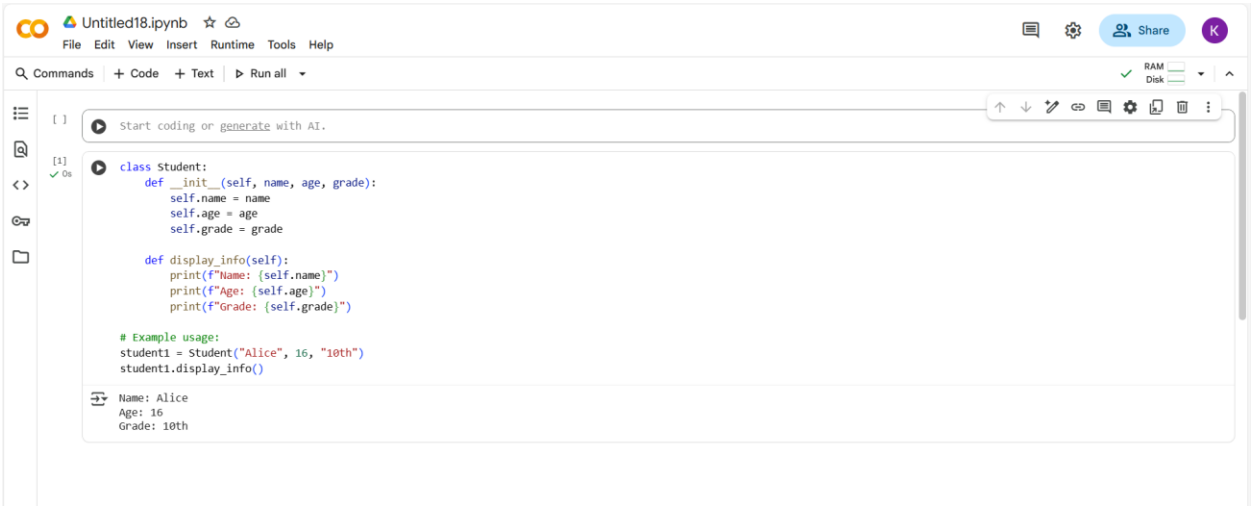
PROMPT:

Write a Python class called Student with attributes like name, age, and grade. Add a method to display the student's information. After generating the code, check the output and analyze the logic for clarity and correctness.

QUESTION:



CODE with OUTPUT:



COMMENT:

Class Definition:

- `class Student:` : This line defines a new class named `Student`. Classes are blueprints for creating objects (like individual students in this case).
- `"""Represents a student with name, age, grade, and hall ticket number."""` : This is a docstring, which explains what the class does.
- `__init__(self, name, age, grade, hall_ticket_number):` : This is the constructor method. It's called when you create a new `Student` object.
  - `self` : Refers to the instance of the class being created.
  - `name`, `age`, `grade`, `hall_ticket_number` : These are parameters that you pass when creating a `Student` object.
  - `self.name = name` : This line assigns the value of the `name` parameter to the `name` attribute of the object. The same applies to `age`, `grade`, and `hall_ticket_number`.

Methods:

- `display_info(self):` : This is a method (a function within a class) that displays the student's information.
  - `self` : Again, refers to the instance of the class.
  - `print(f"...")` : These lines print the student's details using f-strings for easy formatting. They access the attributes using `self.attribute_name`.

User Input and Object Creation:

- `name = input("Enter student name: ")` : This line prompts the user to enter the student's name and stores it in the `name` variable. The same applies to `age`, `grade`, and `hall_ticket_number`. Note that `age` is converted to an integer using `int()`.
- `user_student = Student(name, age, grade, hall_ticket_number)` : This line creates a new `Student` object using the input provided by the user and assigns it to the `user_student` variable.
- `user_student.display_info()` : This line calls the `display_info` method on the `user_student` object to display the information entered by the user.

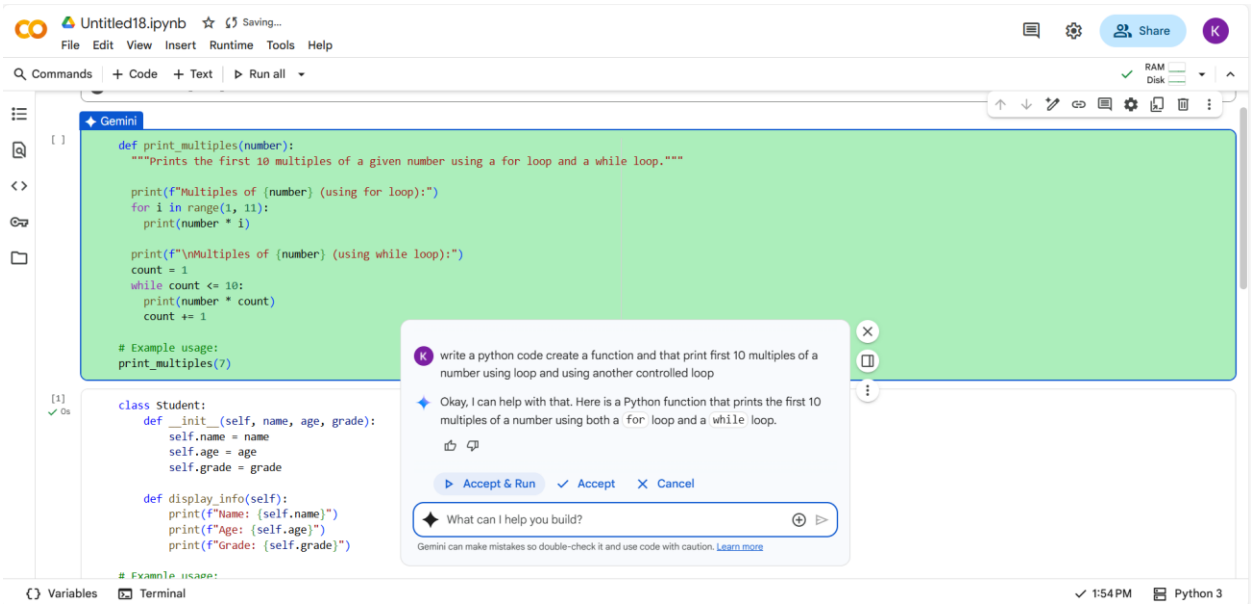
Task Description -2:

Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.
Analyze the generated code
Ask AI to generate code using other controlled looping.

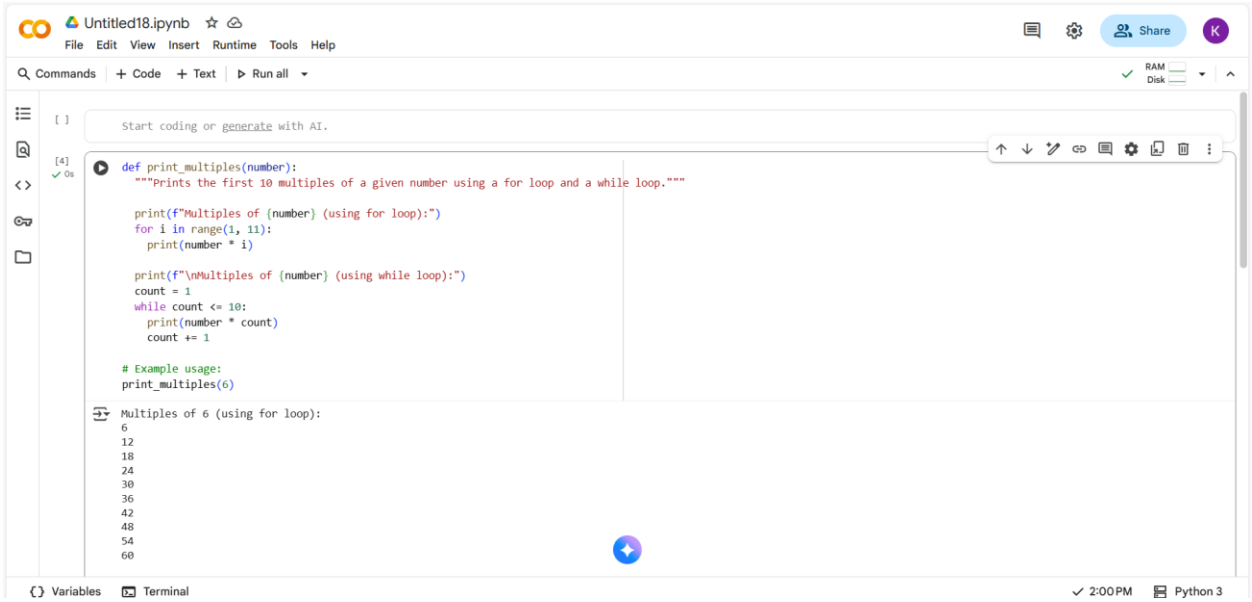
PROMPT:

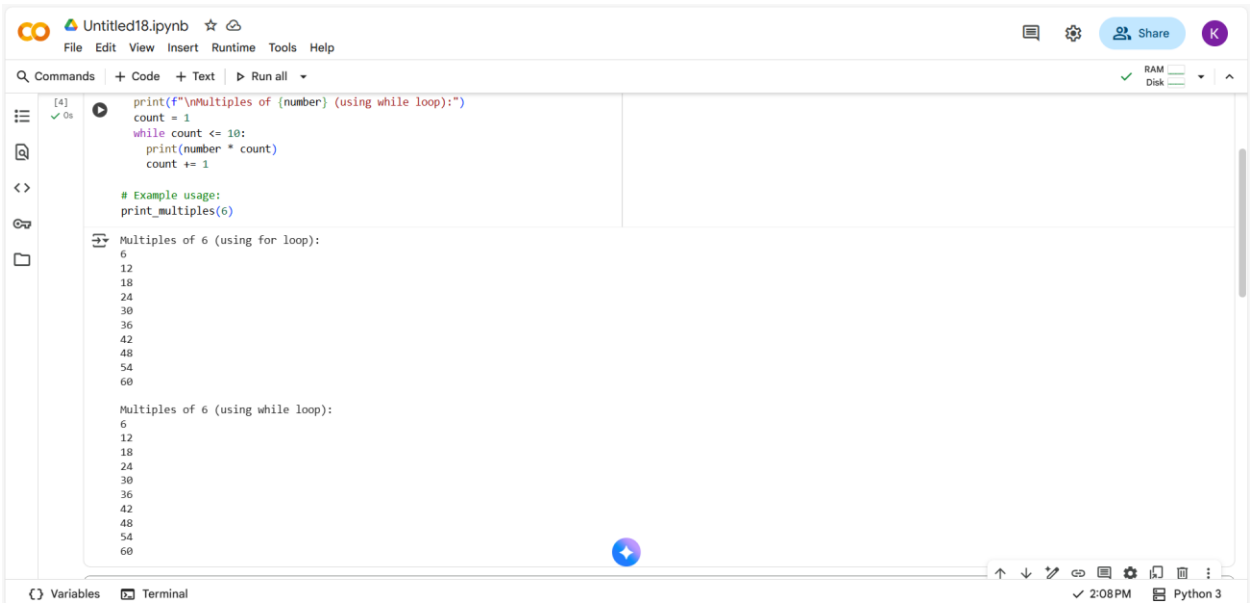
Write a Python function that prints the first 10 multiples of a given number using a loop. Then rewrite the function using a different loop type and compare both versions.

QUESTION:

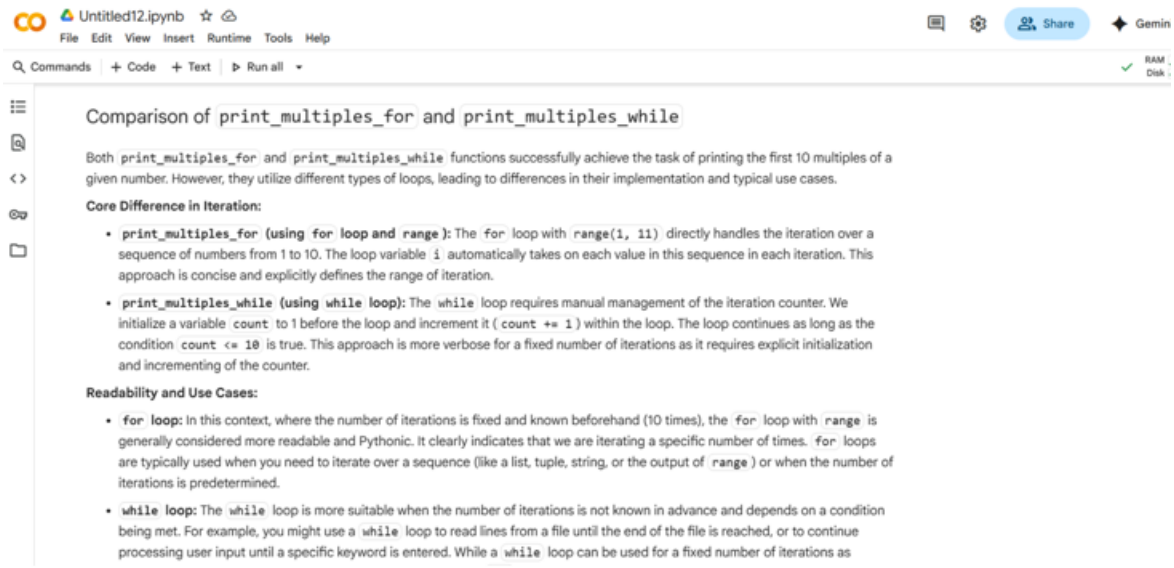


CODE WITH OUTPUT:

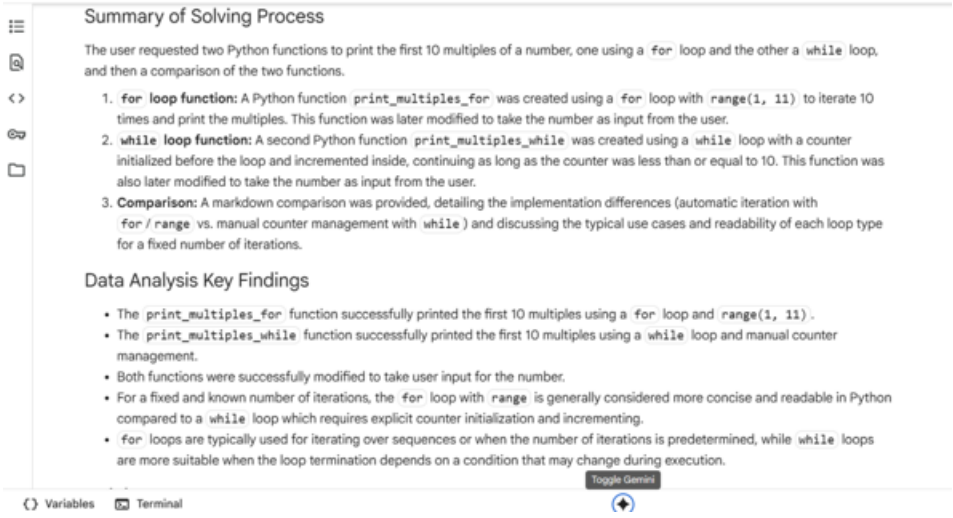



Untitled18.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[4]
0s
print(f"\nMultiples of {number} (using while loop):")
count = 1
while count <= 10:
print(number * count)
count += 1
# Example usage:
print_multiples(6)
Multiples of 6 (using for loop):
6
12
18
24
30
36
42
48
54
60
Multiples of 6 (using while loop):
6
12
18
24
30
36
42
48
54
60
Variables
Terminal
2:08PM
Python 3

COMPARISON:



SUMMARY:



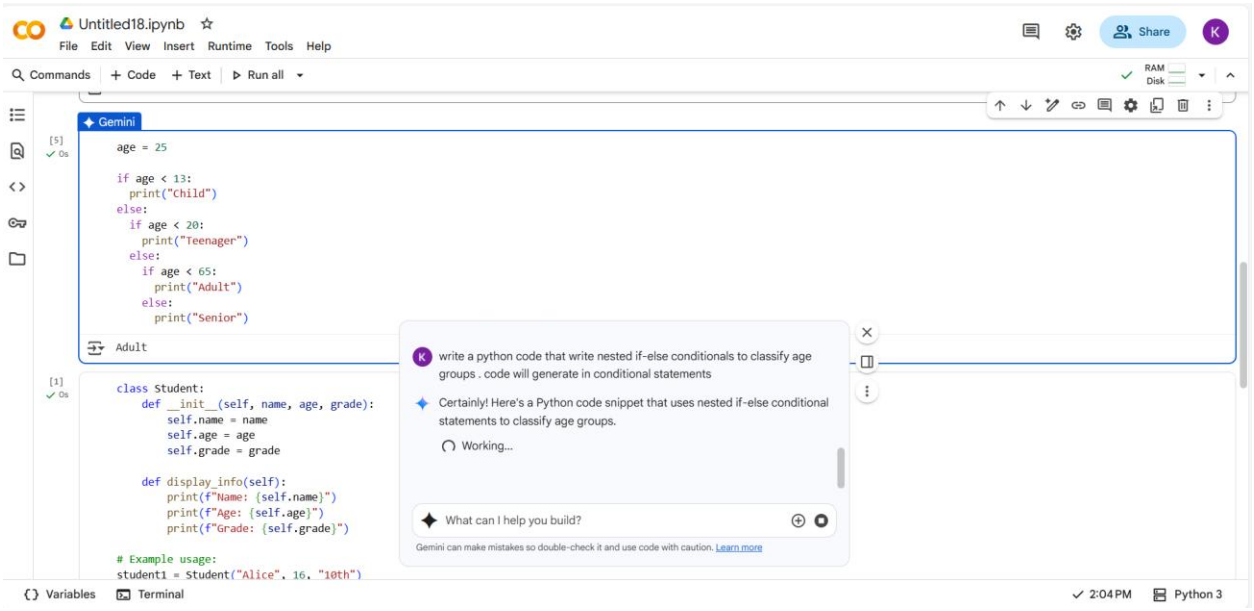
Task Description -3:

Ask AI to write nested if-elif-else conditionals to classify age groups. Analyze the generated code Ask AI to generate code using other conditional statements. write perfect and simple prompt for this question.

PROMPT:

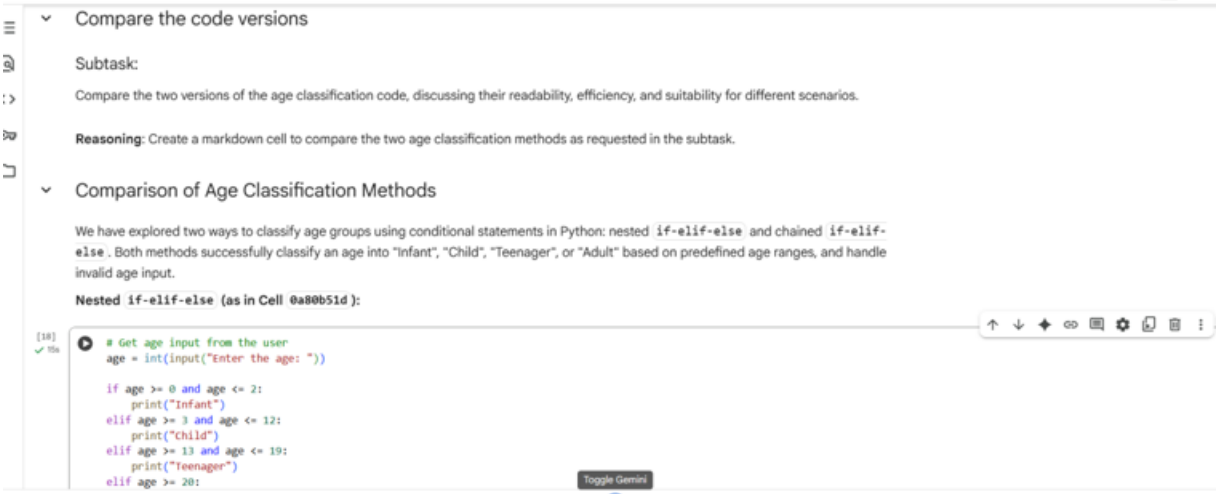
Write Python code using nested if-elif-else statements to classify age groups like infant, child, teenager, and adult. Then rewrite the code using a different conditional method and compare both versions.

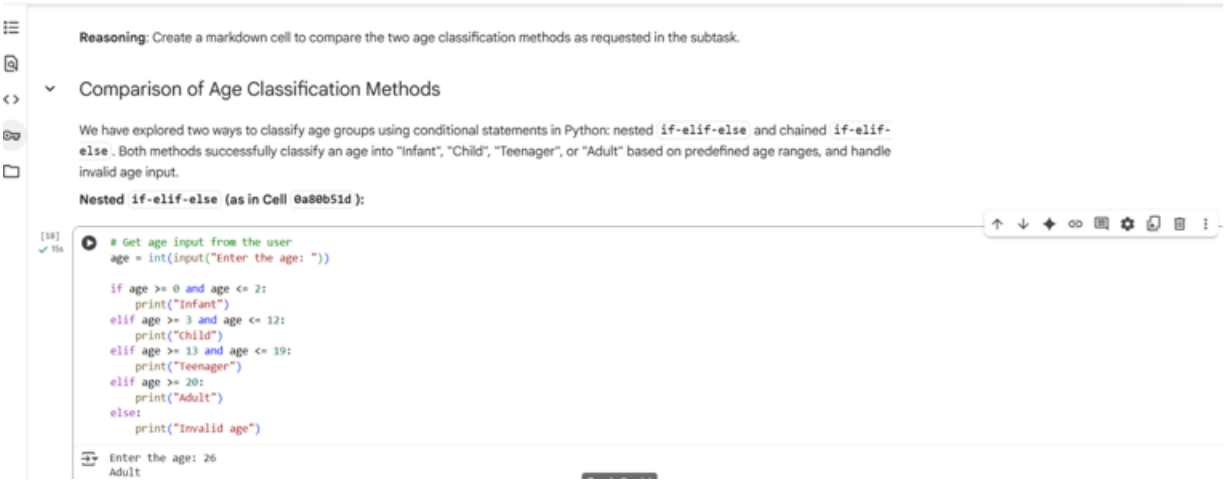
QUESTION:



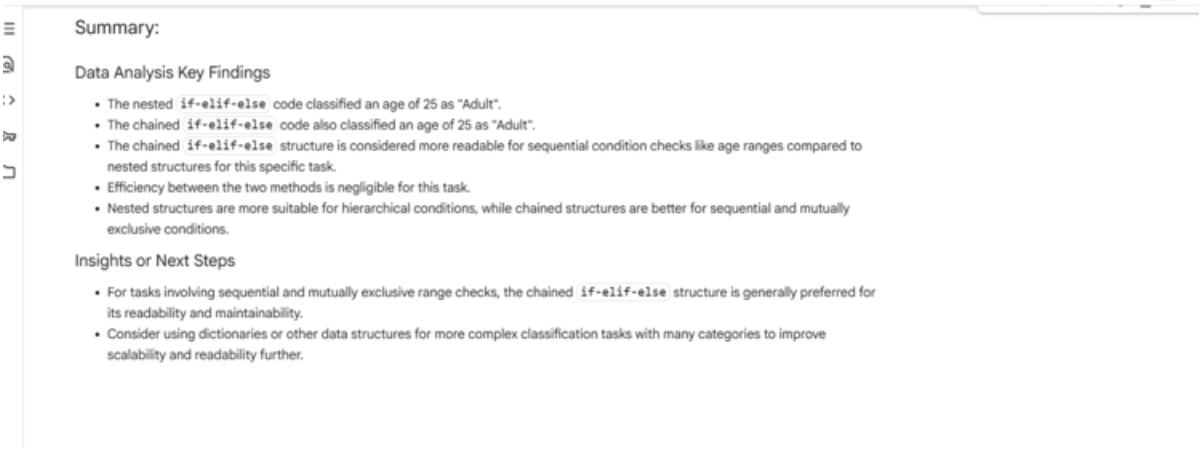
CODE with OUTPUT:


Untitled20.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
# Version 2: Using a series of independent if statements
age = int(input("Enter age: "))
age_group = "Invalid age" # Default to invalid age
if 0 <= age <= 1:
age_group = "Infant"
if 2 <= age <= 12:
age_group = "Child"
if 13 <= age <= 19:
age_group = "Teenager"
if age > 19:
age_group = "Adult"
print(f"Age: {age}, Age Group (independent if statements): {age_group}")
Enter age: 19
Age: 19, Age Group (independent if statements): Teenager


Compare the code versions
Subtask:
Compare the two versions of the age classification code, discussing their readability, efficiency, and suitability for different scenarios.
Reasoning: Create a markdown cell to compare the two age classification methods as requested in the subtask.
Comparison of Age Classification Methods
We have explored two ways to classify age groups using conditional statements in Python: nested if-elif-else and chained if-elif-else. Both methods successfully classify an age into "Infant", "Child", "Teenager", or "Adult" based on predefined age ranges, and handle invalid age input.
Nested if-elif-else (as in Cell 0a80b51d):
# Get age input from the user
age = int(input("Enter the age: "))
if age >= 0 and age <= 2:
print("Infant")
elif age >= 3 and age <= 12:
print("Child")
elif age >= 13 and age <= 19:
print("Teenager")
elif age >= 20:
Toggle Gemini


Reasoning: Create a markdown cell to compare the two age classification methods as requested in the subtask.
Comparison of Age Classification Methods
We have explored two ways to classify age groups using conditional statements in Python: nested if-elif-else and chained if-elif-else. Both methods successfully classify an age into "Infant", "Child", "Teenager", or "Adult" based on predefined age ranges, and handle invalid age input.
Nested if-elif-else (as in Cell 0a80b51d):
# Get age input from the user
age = int(input("Enter the age: "))
if age >= 0 and age <= 2:
print("Infant")
elif age >= 3 and age <= 12:
print("Child")
elif age >= 13 and age <= 19:
print("Teenager")
elif age >= 20:
print("Adult")
else:
print("Invalid age")
Enter the age: 26
Adult

SUMMARY:

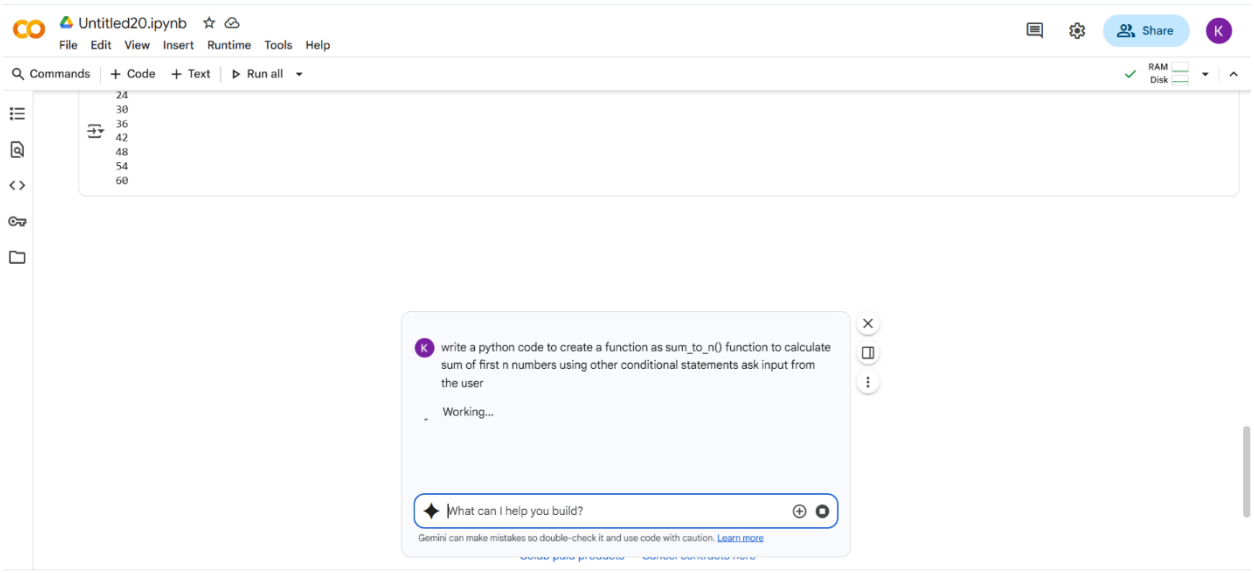


Task Description -4:

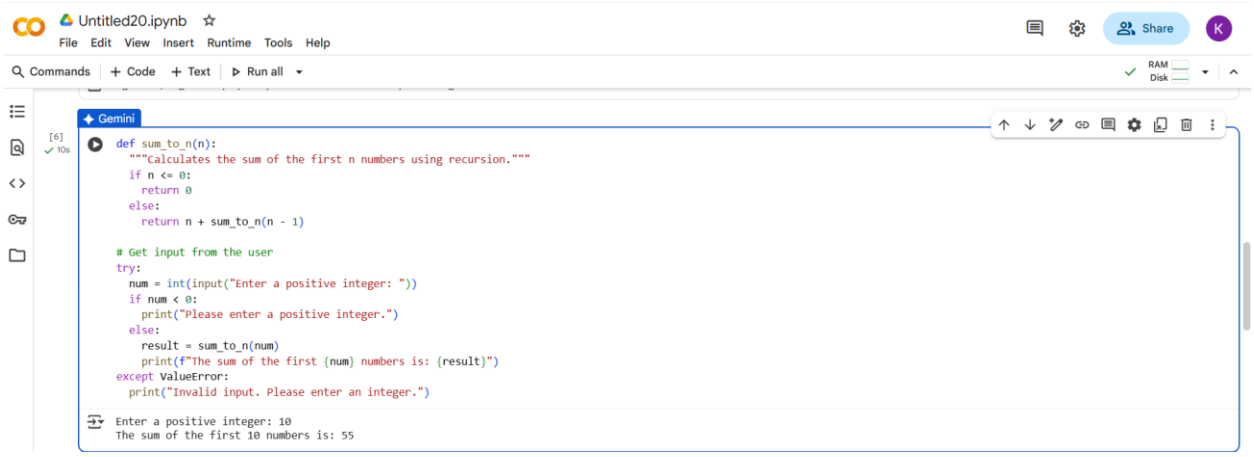
Generate a sum_to_n() function to calculate sum of first n numbers
• Analyze the generated code
• Get suggestions from AI with other controlled looping.

PROMPT:

write a python code to create a function as sum_to_n() function to calculate sum of first n numbers using other conditional statements ask input from the userQUESTION:



CODE WITH OUTPUT:



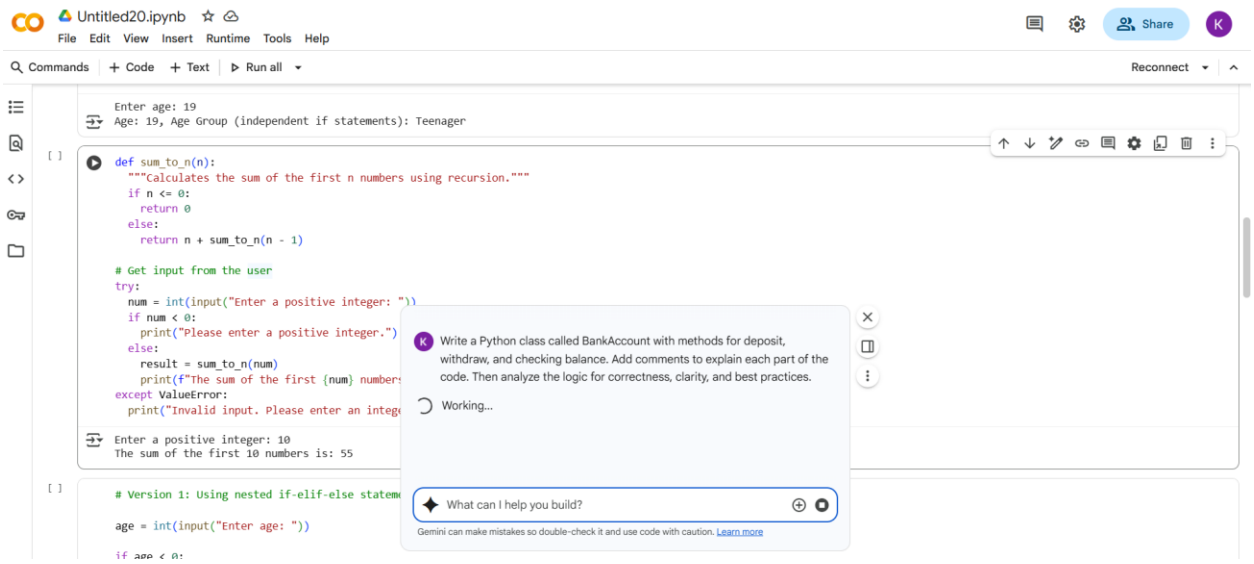
Task Description -5:

Use AI to build a BankAccount class with deposit, withdraw, and balance methods.
• Analyze the generated code
• Add comments and explain code.

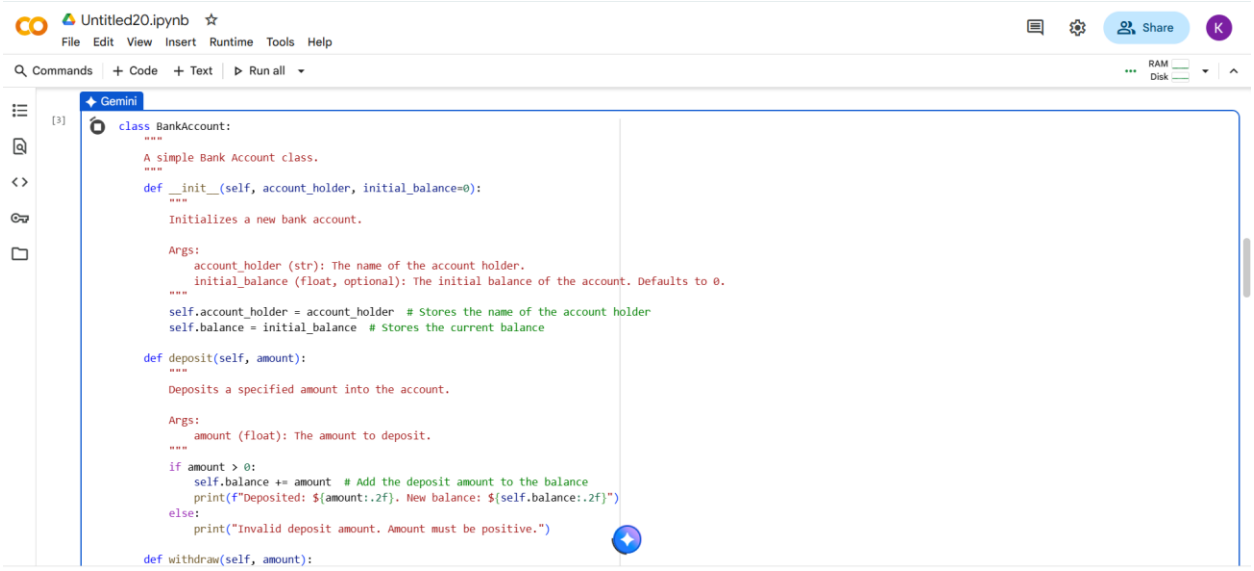
PROMPT:

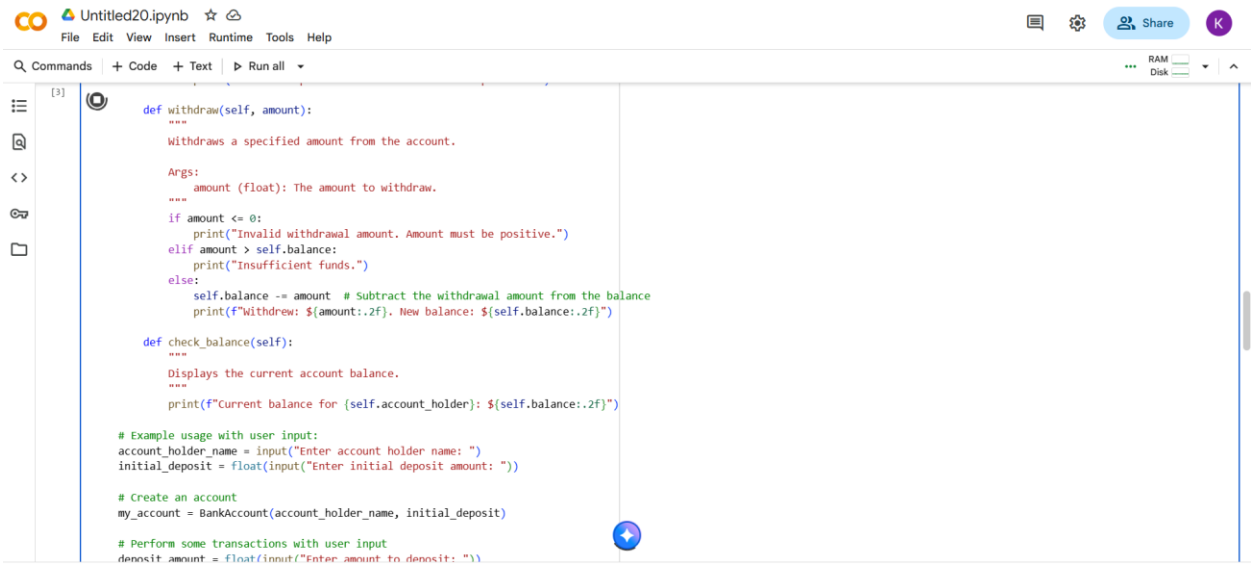
Write a Python class called `BankAccount` with methods for `deposit`, `withdraw`, and checking `balance`. Add comments to explain each part of the code. Then analyze the logic for correctness, clarity, and best practices.

QUESTION:

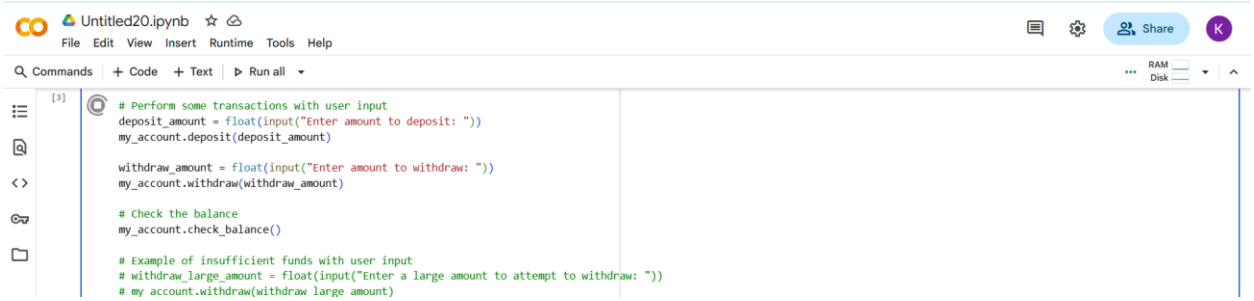

Untitled20.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
Reconnect
Enter age: 19
Age: 19, Age Group (independent if statements): Teenager
def sum_to_n(n):
"""Calculates the sum of the first n numbers using recursion."""
if n <= 0:
return 0
else:
return n + sum_to_n(n - 1)
# Get input from the user
try:
num = int(input("Enter a positive integer: "))
if num < 0:
print("Please enter a positive integer.")
else:
result = sum_to_n(num)
print(f"The sum of the first {num} numbers
except ValueError:
print("Invalid input. Please enter an intege
Enter a positive integer: 10
The sum of the first 10 numbers is: 55
Write a Python class called BankAccount with methods for deposit, withdraw, and checking balance. Add comments to explain each part of the code. Then analyze the logic for correctness, clarity, and best practices.
Working...
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
# Version 1: Using nested if-elif-else statem
age = int(input("Enter age: "))
if age < 0:

CODE:


Untitled20.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
Gemini
class BankAccount:
"""
A simple Bank Account class.
"""
def __init__(self, account_holder, initial_balance=0):
"""
Initializes a new bank account.
Args:
account_holder (str): The name of the account holder.
initial_balance (float, optional): The initial balance of the account. Defaults to 0.
"""
self.account_holder = account_holder # Stores the name of the account holder
self.balance = initial_balance # Stores the current balance
def deposit(self, amount):
"""
Deposits a specified amount into the account.
Args:
amount (float): The amount to deposit.
"""
if amount > 0:
self.balance += amount # Add the deposit amount to the balance
print(f"Deposited: ${amount:.2f}. New balance: ${self.balance:.2f}")
else:
print("Invalid deposit amount. Amount must be positive.")
def withdraw(self, amount):



```
[3]
    def withdraw(self, amount):
        """
        Withdraws a specified amount from the account.

        Args:
            amount (float): The amount to withdraw.
        """
        if amount <= 0:
            print("Invalid withdrawal amount. Amount must be positive.")
        elif amount > self.balance:
            print("Insufficient funds.")
        else:
            self.balance -= amount  # Subtract the withdrawal amount from the balance
            print(f"Withdrew: ${amount:.2f}. New balance: ${self.balance:.2f}")

    def check_balance(self):
        """
        Displays the current account balance.
        """
        print(f"Current balance for {self.account_holder}: ${self.balance:.2f}")

# Example usage with user input:
account_holder_name = input("Enter account holder name: ")
initial_deposit = float(input("Enter initial deposit amount: "))

# Create an account
my_account = BankAccount(account_holder_name, initial_deposit)

# Perform some transactions with user input
deposit_amount = float(input("Enter amount to deposit: "))
```



```
[3]
# Perform some transactions with user input
deposit_amount = float(input("Enter amount to deposit: "))
my_account.deposit(deposit_amount)

withdraw_amount = float(input("Enter amount to withdraw: "))
my_account.withdraw(withdraw_amount)

# Check the balance
my_account.check_balance()

# Example of insufficient funds with user input
# withdraw_large_amount = float(input("Enter a large amount to attempt to withdraw: "))
# my account.withdraw(withdraw large amount)
```

OUTPUT:

```
[3]
1m
my_account.check_balance()

# Example of insufficient funds with user input
# withdraw_large_amount = float(input("Enter a large amount to attempt to withdraw: "))
# my_account.withdraw(withdraw_large_amount)

Enter account holder name: manohar
Enter initial deposit amount: 20000
Enter amount to deposit: 10000
Deposited: $10000.00. New balance: $30000.00
Enter amount to withdraw: 20000
Withdrew: $20000.00. New balance: $10000.00
Current balance for manohar: $10000.00
```

EXPLANATION:

Gemini ×

1. `# Example User Interaction:` : This is a comment indicating that the following lines are an example of how a user would interact with the `BankAccount` class.
2. `# Create a bank account` : Another comment explaining the next line.
3. `my_account = BankAccount(1000)` : This line creates an *instance* of the `BankAccount` class. Think of `my_account` as a specific bank account object. When `BankAccount(1000)` is called, the `__init__` method within the `BankAccount` class is executed, setting the initial balance of this new account to 1000.
4. `# Get user input for deposit` : Comment for the deposit part.
5. `deposit_amount = float(input("Enter amount to deposit: "))` : This line prompts the user to type in a value after displaying the message "Enter amount to deposit: ". The `input()` function reads whatever the user types as a string. `float()` then converts that string into a floating-point number (a number with a decimal point), and this value is stored in the variable `deposit_amount`.
6. `my_account.deposit(deposit_amount)` : This line calls the `deposit` method on the `my_account` object. The value stored in `deposit_amount` (the number the user entered) is passed as the `amount` argument to the `deposit` method.

Gemini ×

7. `# Get user input for withdrawal` : Comment for the withdrawal part.
8. `withdraw_amount = float(input("Enter amount to withdraw: "))` : Similar to the deposit input, this line prompts the user for a withdrawal amount, reads their input as a string, converts it to a floating-point number, and stores it in the `withdraw_amount` variable.
9. `my_account.withdraw(withdraw_amount)` : This line calls the `withdraw` method on the `my_account` object, passing the user's withdrawal amount to the method.
10. `# Check balance` : Comment for the balance check.
11. `print(f"Current balance: ${my_account.get_balance()}")` : This line first calls the `get_balance()` method on the `my_account` object. This method returns the current value of the `balance` attribute. The `print()` function then displays the text "Current balance: $" followed by the balance value returned by `get_balance()`. The `f""` is an f-string, which allows you to embed the value of variables directly within the string.

-----THANK YOU-----