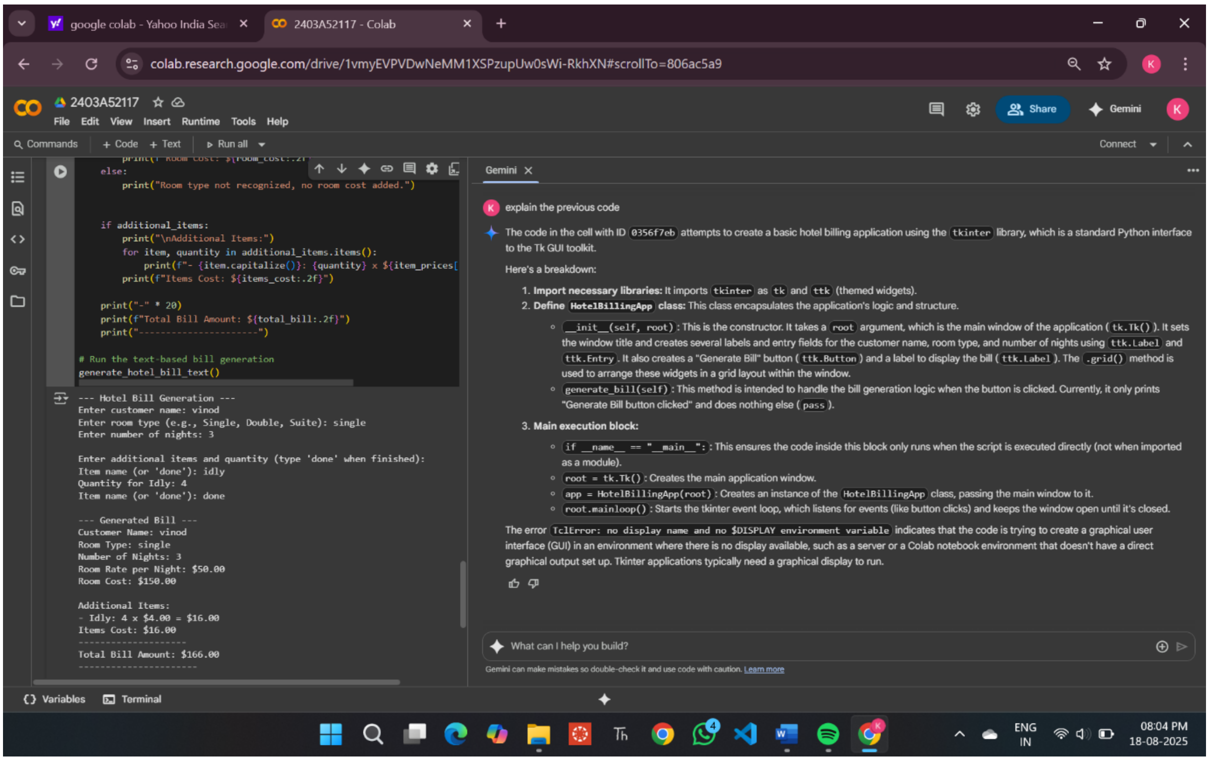

google colab - Yahoo India Sea
2403A52117 - Colab
colab.research.google.com/drive/1vmyEVPVDwNeMM1XSPzupUw0sWi-RkhXN#scrollTo=806ac5a9
2403A52117
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect
Gemini
Variables
Terminal
ENG
IN
08:04 PM
18-08-2025

```
    else:
        print("Room type not recognized, no room cost added.")


    if additional_items:
        print("\nAdditional Items:")
        for item, quantity in additional_items.items():
            print(f"- {item.capitalize()}: {quantity} x ${item_prices[
        print(f"Items Cost: ${items_cost:.2f}")

    print("-" * 20)
    print(f"Total Bill Amount: ${total_bill:.2f}")
    print("--------------------")

# Run the text-based bill generation
generate_hotel_bill_text()
```

```
--- Hotel Bill Generation ---
Enter customer name: vinod
Enter room type (e.g., Single, Double, Suite): single
Enter number of nights: 3

Enter additional items and quantity (type 'done' when finished):
Item name (or 'done'): idly
Quantity for Idly: 4
Item name (or 'done'): done

--- Generated Bill ---
Customer Name: vinod
Room Type: single
Number of Nights: 3
Room Rate per Night: $50.00
Room Cost: $150.00

Additional Items:
- Idly: 4 x $4.00 = $16.00
Items Cost: $16.00
--------------------
Total Bill Amount: $166.00
--------------------
```

explain the previous code

The code in the cell with ID `0356f7eb` attempts to create a basic hotel billing application using the `tkinter` library, which is a standard Python interface to the Tk GUI toolkit.

Here's a breakdown:

1. **Import necessary libraries:** It imports `tkinter` as `tk` and `ttk` (themed widgets).
2. **Define `HotelBillingApp` class:** This class encapsulates the application's logic and structure.
    - `__init__(self, root)`: This is the constructor. It takes a `root` argument, which is the main window of the application (`tk.Tk()`). It sets the window title and creates several labels and entry fields for the customer name, room type, and number of nights using `ttk.Label` and `ttk.Entry`. It also creates a "Generate Bill" button (`ttk.Button`) and a label to display the bill (`ttk.Label`). The `.grid()` method is used to arrange these widgets in a grid layout within the window.
    - `generate_bill(self)`: This method is intended to handle the bill generation logic when the button is clicked. Currently, it only prints "Generate Bill button clicked" and does nothing else (`pass`).
3. **Main execution block:**
    - `if __name__ == "__main__":`: This ensures the code inside this block only runs when the script is executed directly (not when imported as a module).
    - `root = tk.Tk()`: Creates the main application window.
    - `app = HotelBillingApp(root)`: Creates an instance of the `HotelBillingApp` class, passing the main window to it.
    - `root.mainloop()`: Starts the tkinter event loop, which listens for events (like button clicks) and keeps the window open until it's closed.

The error `TclError: no display name and no $DISPLAY environment variable` indicates that the code is trying to create a graphical user interface (GUI) in an environment where there is no display available, such as a server or a Colab notebook environment that doesn't have a direct graphical output set up. Tkinter applications typically need a graphical display to run.

What can I help you build?

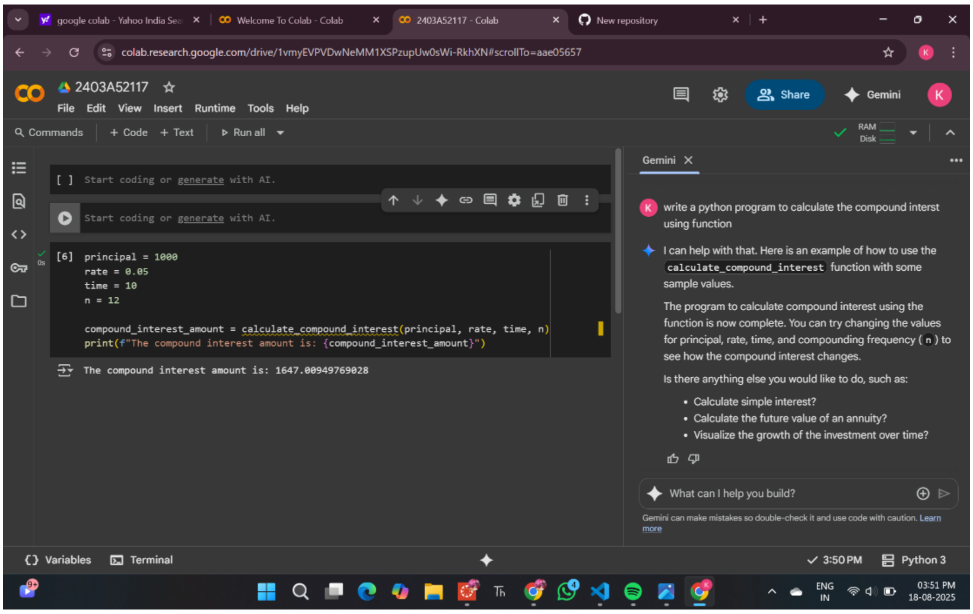
Gemini can make mistakes so double-check it and use code with caution. Learn more

ASSIGNMENT – 3.2

NAME : K.SHARATHCHANDRA

HT.NO : 2403A52117

TASK 1:



```
principal = 1000
rate = 0.05
time = 10
n = 12

compound_interest_amount = calculate_compound_interest(principal, rate, time, n)
print(f"The compound interest amount is: {compound_interest_amount}")
```

```
The compound interest amount is: 1647.00949769028
```

write a python program to calculate the compound interest using function

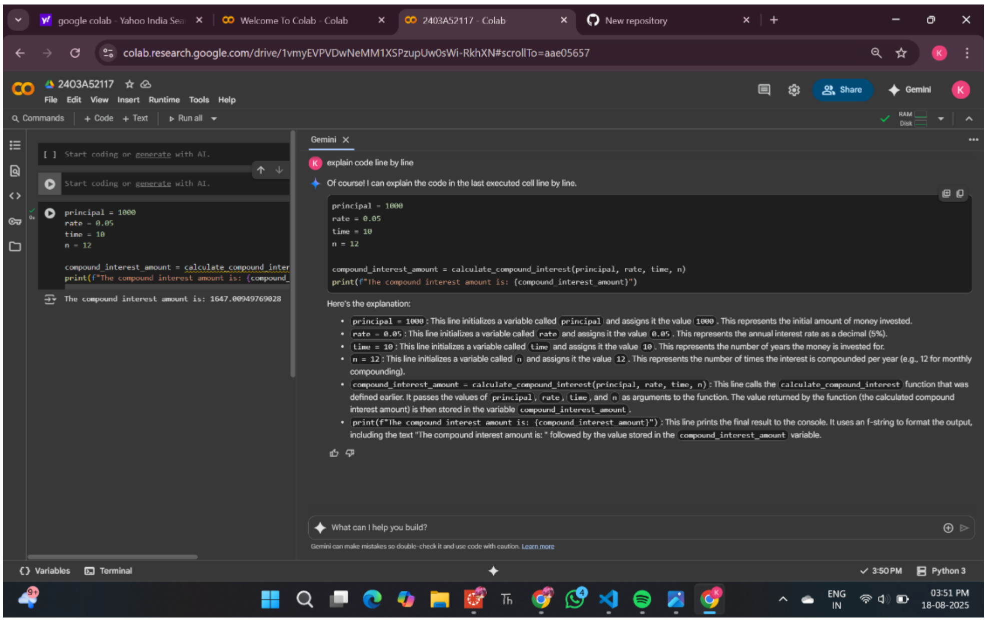
I can help with that. Here is an example of how to use the `calculate_compound_interest` function with some sample values.

The program to calculate compound interest using the function is now complete. You can try changing the values for principal, rate, time, and compounding frequency (`n`) to see how the compound interest changes.

Is there anything else you would like to do, such as:

- Calculate simple interest?
- Calculate the future value of an annuity?
- Visualize the growth of the investment over time?

EXPLANATION :



explain code line by line

Of course! I can explain the code in the last executed cell line by line.

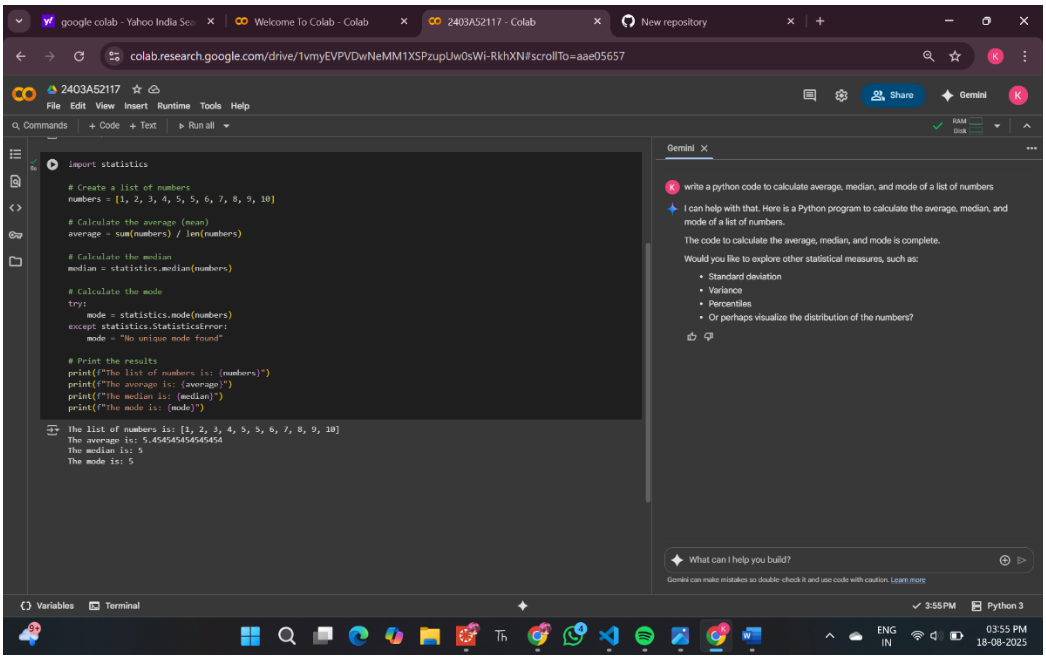
```
principal = 1000
rate = 0.05
time = 10
n = 12

compound_interest_amount = calculate_compound_interest(principal, rate, time, n)
print(f"The compound interest amount is: {compound_interest_amount}")
```

Here's the explanation:

- `principal = 1000`: This line initializes a variable called `principal` and assigns it the value `1000`. This represents the initial amount of money invested.
- `rate = 0.05`: This line initializes a variable called `rate` and assigns it the value `0.05`. This represents the annual interest rate as a decimal (5%).
- `time = 10`: This line initializes a variable called `time` and assigns it the value `10`. This represents the number of years the money is invested for.
- `n = 12`: This line initializes a variable called `n` and assigns it the value `12`. This represents the number of times the interest is compounded per year (e.g., 12 for monthly compounding).
- `compound_interest_amount = calculate_compound_interest(principal, rate, time, n)`: This line calls the `calculate_compound_interest` function that was defined earlier. It passes the values of `principal`, `rate`, `time`, and `n` as arguments to the function. The value returned by the function (the calculated compound interest amount) is then stored in the variable `compound_interest_amount`.
- `print(f"The compound interest amount is: {compound_interest_amount}")`: This line prints the final result to the console. It uses an f-string to format the output, including the text "The compound interest amount is: " followed by the value stored in the `compound_interest_amount` variable.

TASK 2:


google colab - Yahoo India Sea
Welcome To Colab - Colab
2403A52117 - Colab
New repository
colab.research.google.com/drive/1vmyEVPVDwNeMM1XSPzupUw0sWi-RkhXN#scrollTo=aae05657
2403A52117
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
RAM Disk
Gemini
What can I help you build?
Variables Terminal
3:55 PM Python 3
ENG IN
03:55 PM 18-08-2025

```python
import statistics

# Create a list of numbers
numbers = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]

# Calculate the average (mean)
average = sum(numbers) / len(numbers)

# Calculate the median
median = statistics.median(numbers)

# Calculate the mode
try:
    mode = statistics.mode(numbers)
except statistics.StatisticsError:
    mode = "No unique mode found"

# Print the results
print(f"The list of numbers is: {numbers}")
print(f"The average is: {average}")
print(f"The median is: {median}")
print(f"The mode is: {mode}")
```

```
The list of numbers is: [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
The average is: 5.454545454545454
The median is: 5
The mode is: 5
```

write a python code to calculate average, median, and mode of a list of numbers

I can help with that. Here is a Python program to calculate the average, median, and mode of a list of numbers.

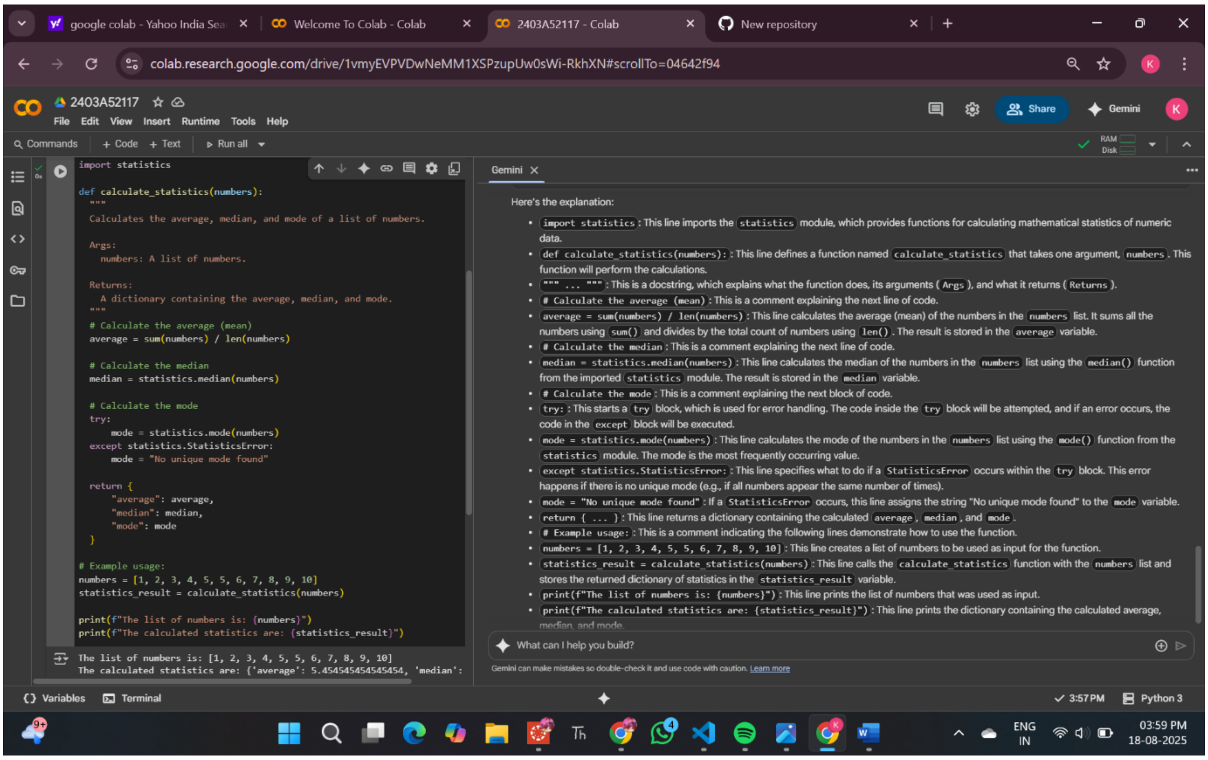
The code to calculate the average, median, and mode is complete.

Would you like to explore other statistical measures, such as:

- Standard deviation
- Variance
- Percentiles
- Or perhaps visualize the distribution of the numbers?

Gemini can make mistakes so double-check it and use code with caution. Learn more

EXPLANATION:


google colab - Yahoo India Sea
Welcome To Colab - Colab
2403A52117 - Colab
New repository
colab.research.google.com/drive/1vmyEVPVDwNeMM1XSPzupUw0sWi-RkhXN#scrollTo=04642f94
2403A52117
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
RAM Disk
Gemini
What can I help you build?
Variables Terminal
3:57 PM Python 3
ENG IN
03:59 PM 18-08-2025

```python
import statistics

def calculate_statistics(numbers):
  """
  Calculates the average, median, and mode of a list of numbers.

  Args:
    numbers: A list of numbers.

  Returns:
    A dictionary containing the average, median, and mode.
  """
  # Calculate the average (mean)
  average = sum(numbers) / len(numbers)

  # Calculate the median
  median = statistics.median(numbers)

  # Calculate the mode
  try:
      mode = statistics.mode(numbers)
  except statistics.StatisticsError:
      mode = "No unique mode found"

  return {
      "average": average,
      "median": median,
      "mode": mode
  }

# Example usage:
numbers = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
statistics_result = calculate_statistics(numbers)

print(f"The list of numbers is: {numbers}")
print(f"The calculated statistics are: {statistics_result}")
```

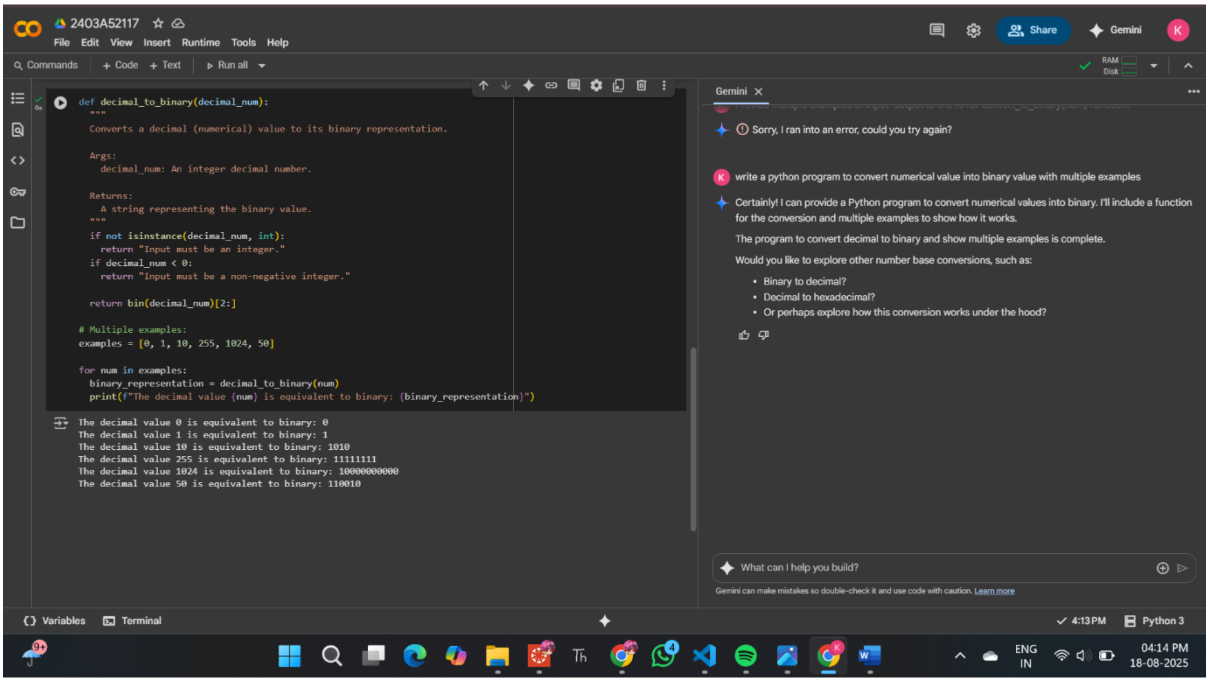
```
The list of numbers is: [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
The calculated statistics are: {'average': 5.454545454545454, 'median':
```

Here's the explanation:

- `import statistics`: This line imports the `statistics` module, which provides functions for calculating mathematical statistics of numeric data.
- `def calculate_statistics(numbers):`: This line defines a function named `calculate_statistics` that takes one argument, `numbers`. This function will perform the calculations.
- `""" ... """`: This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
- `# Calculate the average (mean)`: This is a comment explaining the next line of code.
- `average = sum(numbers) / len(numbers)`: This line calculates the average (mean) of the numbers in the `numbers` list. It sums all the numbers using `sum()` and divides by the total count of numbers using `len()`. The result is stored in the `average` variable.
- `# Calculate the median`: This is a comment explaining the next line of code.
- `median = statistics.median(numbers)`: This line calculates the median of the numbers in the `numbers` list using the `median()` function from the imported `statistics` module. The result is stored in the `median` variable.
- `# Calculate the mode`: This is a comment explaining the next block of code.
- `try:`: This starts a `try` block, which is used for error handling. The code inside the `try` block will be attempted, and if an error occurs, the code in the `except` block will be executed.
- `mode = statistics.mode(numbers)`: This line calculates the mode of the numbers in the `numbers` list using the `mode()` function from the `statistics` module. The mode is the most frequently occurring value.
- `except statistics.StatisticsError:`: This line specifies what to do if a `StatisticsError` occurs within the `try` block. This error happens if there is no unique mode (e.g., if all numbers appear the same number of times).
- `mode = "No unique mode found"`: If a `StatisticsError` occurs, this line assigns the string "No unique mode found" to the `mode` variable.
- `return { ... }`: This line returns a dictionary containing the calculated `average`, `median`, and `mode`.
- `# Example usage:`: This is a comment indicating the following lines demonstrate how to use the function.
- `numbers = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]`: This line creates a list of numbers to be used as input for the function.
- `statistics_result = calculate_statistics(numbers)`: This line calls the `calculate_statistics` function with the `numbers` list and stores the returned dictionary of statistics in the `statistics_result` variable.
- `print(f"The list of numbers is: {numbers}")`: This line prints the list of numbers that was used as input.
- `print(f"The calculated statistics are: {statistics_result}")`: This line prints the dictionary containing the calculated average, median, and mode.

Gemini can make mistakes so double-check it and use code with caution. Learn more

TASK 3:



```python
def decimal_to_binary(decimal_num):
  """
  Converts a decimal (numerical) value to its binary representation.

  Args:
    decimal_num: An integer decimal number.

  Returns:
    A string representing the binary value.
  """
  if not isinstance(decimal_num, int):
    return "Input must be an integer."
  if decimal_num < 0:
    return "Input must be a non-negative integer."

  return bin(decimal_num)[2:]

# Multiple examples:
examples = [0, 1, 10, 255, 1024, 50]

for num in examples:
  binary_representation = decimal_to_binary(num)
  print(f"The decimal value {num} is equivalent to binary: {binary_representation}")
```

```
The decimal value 0 is equivalent to binary: 0
The decimal value 1 is equivalent to binary: 1
The decimal value 10 is equivalent to binary: 1010
The decimal value 255 is equivalent to binary: 11111111
The decimal value 1024 is equivalent to binary: 10000000000
The decimal value 50 is equivalent to binary: 110010
```

Sorry, I ran into an error, could you try again?

write a python program to convert numerical value into binary value with multiple examples

Certainly! I can provide a Python program to convert numerical values into binary. I'll include a function for the conversion and multiple examples to show how it works.

The program to convert decimal to binary and show multiple examples is complete.

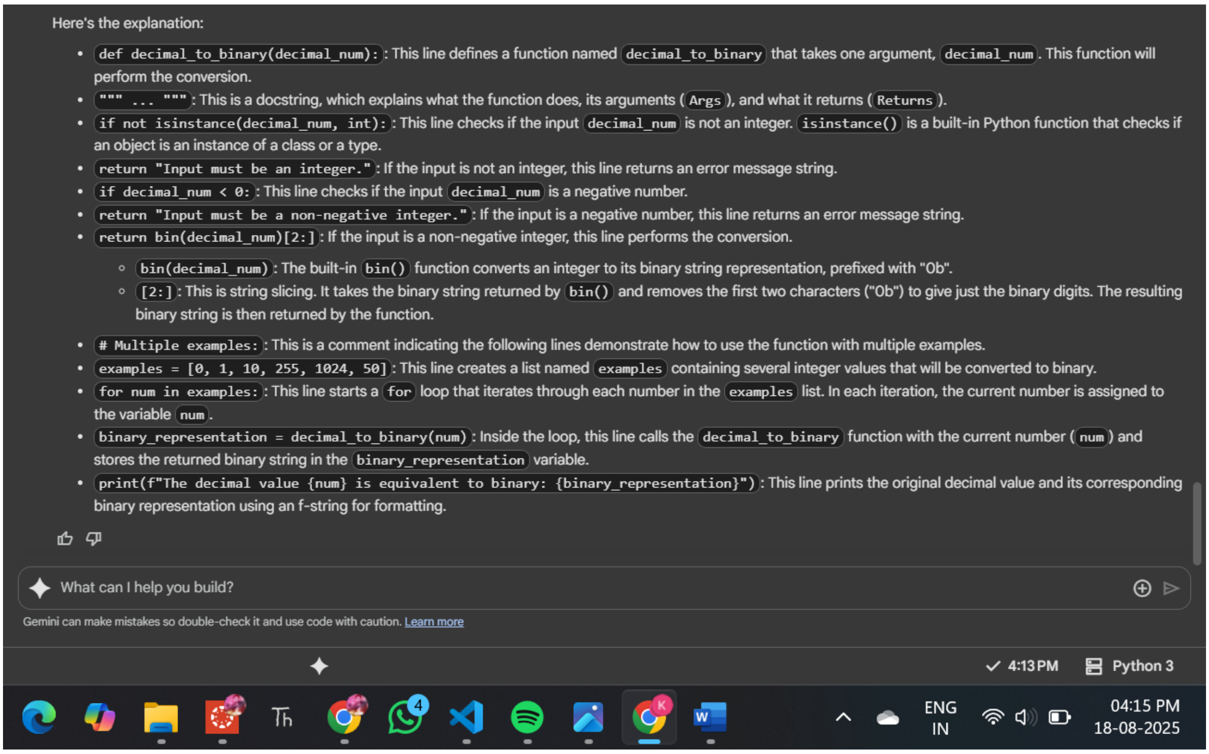
Would you like to explore other number base conversions, such as:

- Binary to decimal?
- Decimal to hexadecimal?
- Or perhaps explore how this conversion works under the hood?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

EXPLANATION:



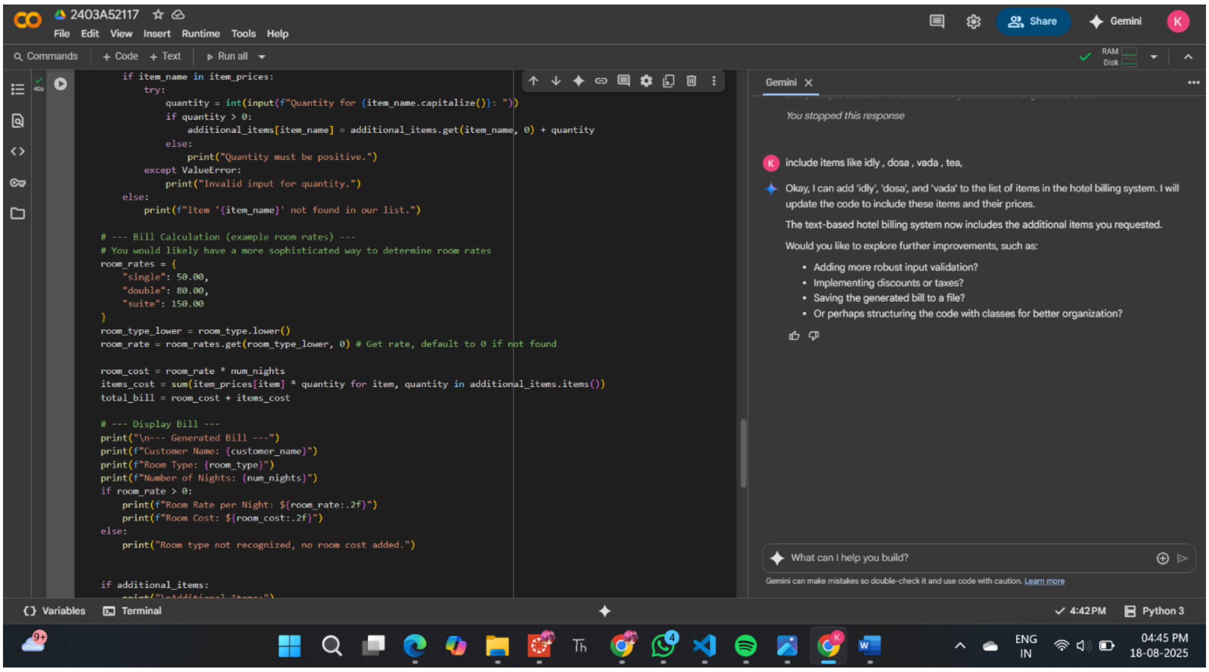
Here's the explanation:

- `def decimal_to_binary(decimal_num):`: This line defines a function named `decimal_to_binary` that takes one argument, `decimal_num`. This function will perform the conversion.
- `""" ... """`: This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
- `if not isinstance(decimal_num, int):`: This line checks if the input `decimal_num` is not an integer. `isinstance()` is a built-in Python function that checks if an object is an instance of a class or a type.
- `return "Input must be an integer."`: If the input is not an integer, this line returns an error message string.
- `if decimal_num < 0:`: This line checks if the input `decimal_num` is a negative number.
- `return "Input must be a non-negative integer."`: If the input is a negative number, this line returns an error message string.
- `return bin(decimal_num)[2:]`: If the input is a non-negative integer, this line performs the conversion.
    - `bin(decimal_num)`: The built-in `bin()` function converts an integer to its binary string representation, prefixed with "0b".
    - `[2:]`: This is string slicing. It takes the binary string returned by `bin()` and removes the first two characters ("0b") to give just the binary digits. The resulting binary string is then returned by the function.
- `# Multiple examples:`: This is a comment indicating the following lines demonstrate how to use the function with multiple examples.
- `examples = [0, 1, 10, 255, 1024, 50]`: This line creates a list named `examples` containing several integer values that will be converted to binary.
- `for num in examples:`: This line starts a `for` loop that iterates through each number in the `examples` list. In each iteration, the current number is assigned to the variable `num`.
- `binary_representation = decimal_to_binary(num)`: Inside the loop, this line calls the `decimal_to_binary` function with the current number (`num`) and stores the returned binary string in the `binary_representation` variable.
- `print(f"The decimal value {num} is equivalent to binary: {binary_representation}")`: This line prints the original decimal value and its corresponding binary representation using an f-string for formatting.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

TASK 4:



```
        if item_name in item_prices:
            try:
                quantity = int(input(f"Quantity for {item_name.capitalize()}: "))
                if quantity > 0:
                    additional_items[item_name] = additional_items.get(item_name, 0) + quantity
                else:
                    print("Quantity must be positive.")
            except ValueError:
                print("Invalid input for quantity.")
        else:
            print(f"Item '{item_name}' not found in our list.")

    # --- Bill Calculation (example room rates) ---
    # You would likely have a more sophisticated way to determine room rates
    room_rates = {
        "single": 50.00,
        "double": 80.00,
        "suite": 150.00
    }
    room_type_lower = room_type.lower()
    room_rate = room_rates.get(room_type_lower, 0) # Get rate, default to 0 if not found

    room_cost = room_rate * num_nights
    items_cost = sum(item_prices[item] * quantity for item, quantity in additional_items.items())
    total_bill = room_cost + items_cost

    # --- Display Bill ---
    print("\n--- Generated Bill ---")
    print(f"Customer Name: {customer_name}")
    print(f"Room Type: {room_type}")
    print(f"Number of Nights: {num_nights}")
    if room_rate > 0:
        print(f"Room Rate per Night: ${room_rate:.2f}")
        print(f"Room Cost: ${room_cost:.2f}")
    else:
        print("Room type not recognized, no room cost added.")


    if additional_items:
```

Gemini

You stopped this response

include items like idly , dosa , vada , tea,

Okay, I can add 'idly', 'dosa', and 'vada' to the list of items in the hotel billing system. I will update the code to include these items and their prices.

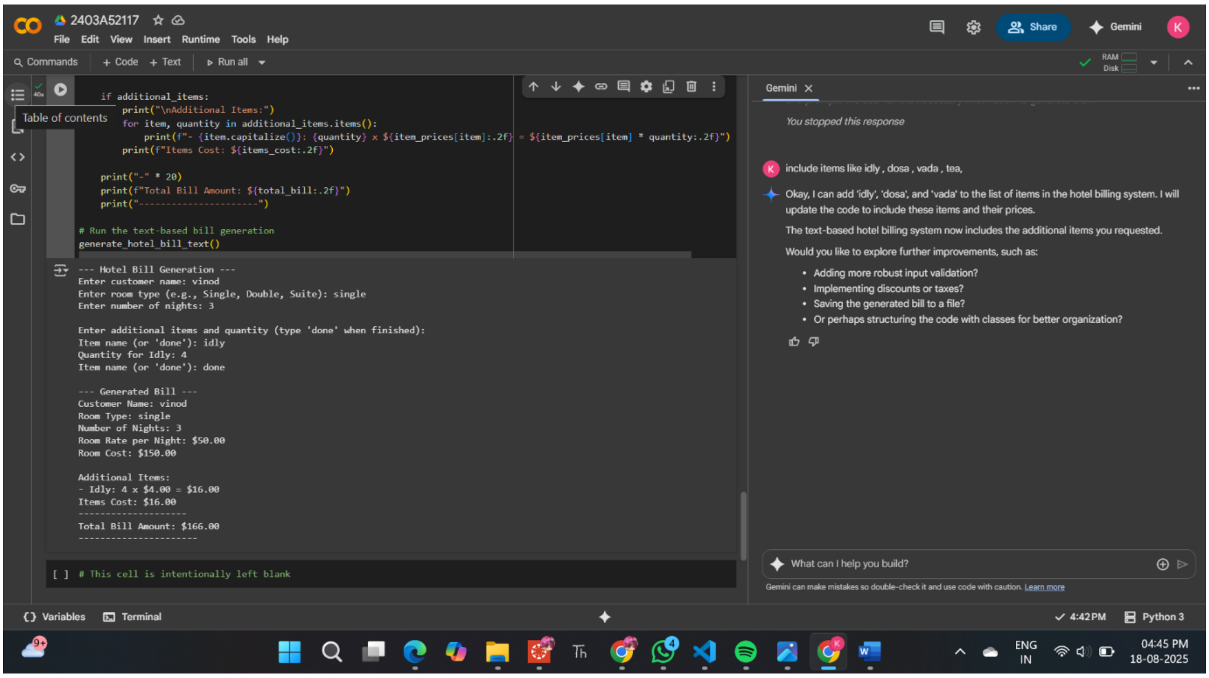
The text-based hotel billing system now includes the additional items you requested.

Would you like to explore further improvements, such as:

- Adding more robust input validation?
- Implementing discounts or taxes?
- Saving the generated bill to a file?
- Or perhaps structuring the code with classes for better organization?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



```
    if additional_items:
        print("\nAdditional Items:")
        for item, quantity in additional_items.items():
            print(f"- {item.capitalize()}: {quantity} x ${item_prices[item]:.2f} = ${item_prices[item] * quantity:.2f}")
        print(f"Items Cost: ${items_cost:.2f}")

    print("-" * 20)
    print(f"Total Bill Amount: ${total_bill:.2f}")
    print("--------------------")

# Run the text-based bill generation
generate_hotel_bill_text()
```

```
--- Hotel Bill Generation ---
Enter customer name: vinod
Enter room type (e.g., Single, Double, Suite): single
Enter number of nights: 3

Enter additional items and quantity (type 'done' when finished):
Item name (or 'done'): idly
Quantity for Idly: 4
Item name (or 'done'): done

--- Generated Bill ---
Customer Name: vinod
Room Type: single
Number of Nights: 3
Room Rate per Night: $50.00
Room Cost: $150.00

Additional Items:
- Idly: 4 x $4.00 = $16.00
Items Cost: $16.00
--------------------
Total Bill Amount: $166.00
--------------------
```

```
[ ] # This cell is intentionally left blank
```

Gemini

You stopped this response

include items like idly , dosa , vada , tea,

Okay, I can add 'idly', 'dosa', and 'vada' to the list of items in the hotel billing system. I will update the code to include these items and their prices.

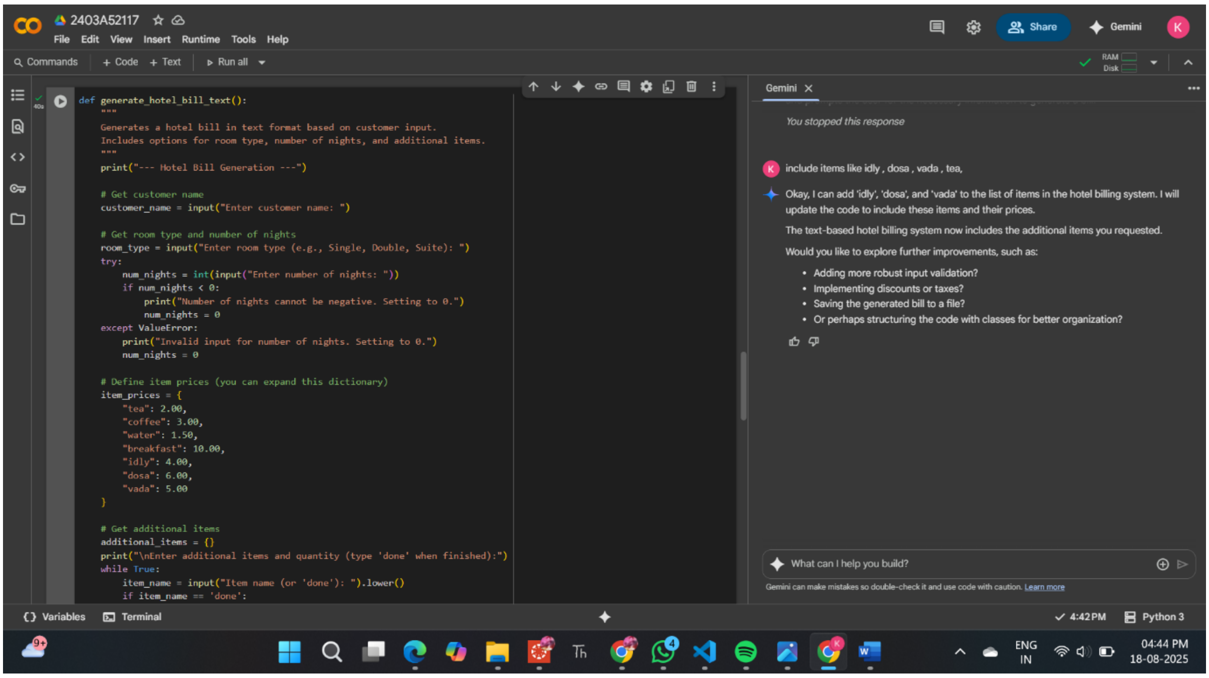
The text-based hotel billing system now includes the additional items you requested.

Would you like to explore further improvements, such as:

- Adding more robust input validation?
- Implementing discounts or taxes?
- Saving the generated bill to a file?
- Or perhaps structuring the code with classes for better organization?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



```
def generate_hotel_bill_text():
    """
    Generates a hotel bill in text format based on customer input.
    Includes options for room type, number of nights, and additional items.
    """
    print("--- Hotel Bill Generation ---")

    # Get customer name
    customer_name = input("Enter customer name: ")

    # Get room type and number of nights
    room_type = input("Enter room type (e.g., Single, Double, Suite): ")
    try:
        num_nights = int(input("Enter number of nights: "))
        if num_nights < 0:
            print("Number of nights cannot be negative. Setting to 0.")
            num_nights = 0
    except ValueError:
        print("Invalid input for number of nights. Setting to 0.")
        num_nights = 0

    # Define item prices (you can expand this dictionary)
    item_prices = {
        "tea": 2.00,
        "coffee": 3.00,
        "water": 1.50,
        "breakfast": 10.00,
        "idly": 4.00,
        "dosa": 6.00,
        "vada": 5.00
    }

    # Get additional items
    additional_items = {}
    print("\nEnter additional items and quantity (type 'done' when finished):")
    while True:
        item_name = input("Item name (or 'done'): ").lower()
        if item_name == 'done':
```

Gemini

You stopped this response

include items like idly , dosa , vada , tea,

Okay, I can add 'idly', 'dosa', and 'vada' to the list of items in the hotel billing system. I will update the code to include these items and their prices.

The text-based hotel billing system now includes the additional items you requested.

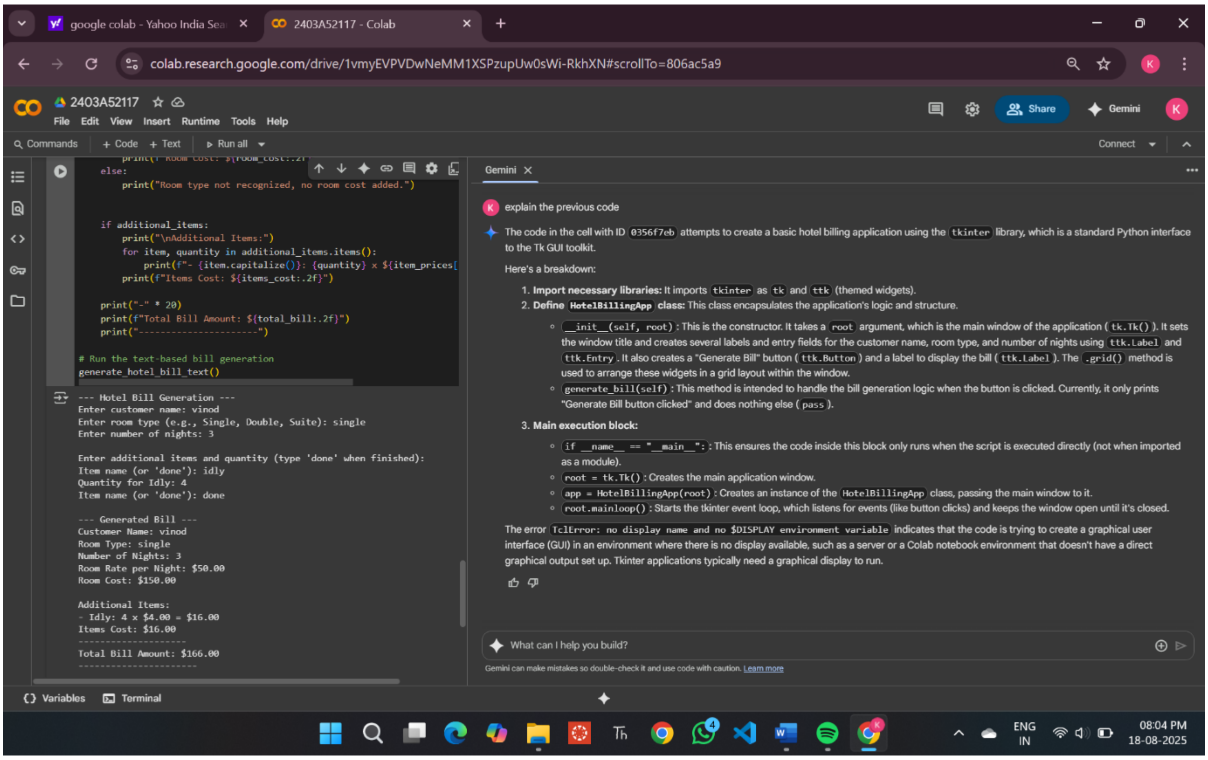
Would you like to explore further improvements, such as:

- Adding more robust input validation?
- Implementing discounts or taxes?
- Saving the generated bill to a file?
- Or perhaps structuring the code with classes for better organization?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

EXPLANATION:


google colab - Yahoo India Sea
2403A52117 - Colab
colab.research.google.com/drive/1vmyEVPVDwNeMM1XSPzupUw0sWi-RkhXN#scrollTo=806ac5a9
2403A52117
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
Connect
Variables Terminal
08:04 PM
18-08-2025

```
    else:
        print("Room type not recognized, no room cost added.")


    if additional_items:
        print("\nAdditional Items:")
        for item, quantity in additional_items.items():
            print(f"- {item.capitalize()}: {quantity} x ${item_prices[
        print(f"Items Cost: ${items_cost:.2f}")

    print("-" * 20)
    print(f"Total Bill Amount: ${total_bill:.2f}")
    print("--------------------")

# Run the text-based bill generation
generate_hotel_bill_text()
```

```
--- Hotel Bill Generation ---
Enter customer name: vinod
Enter room type (e.g., Single, Double, Suite): single
Enter number of nights: 3

Enter additional items and quantity (type 'done' when finished):
Item name (or 'done'): idly
Quantity for Idly: 4
Item name (or 'done'): done

--- Generated Bill ---
Customer Name: vinod
Room Type: single
Number of Nights: 3
Room Rate per Night: $50.00
Room Cost: $150.00

Additional Items:
- Idly: 4 x $4.00 = $16.00
Items Cost: $16.00
--------------------
Total Bill Amount: $166.00
--------------------
```

Gemini

explain the previous code

The code in the cell with ID `0356f7eb` attempts to create a basic hotel billing application using the `tkinter` library, which is a standard Python interface to the Tk GUI toolkit.

Here's a breakdown:

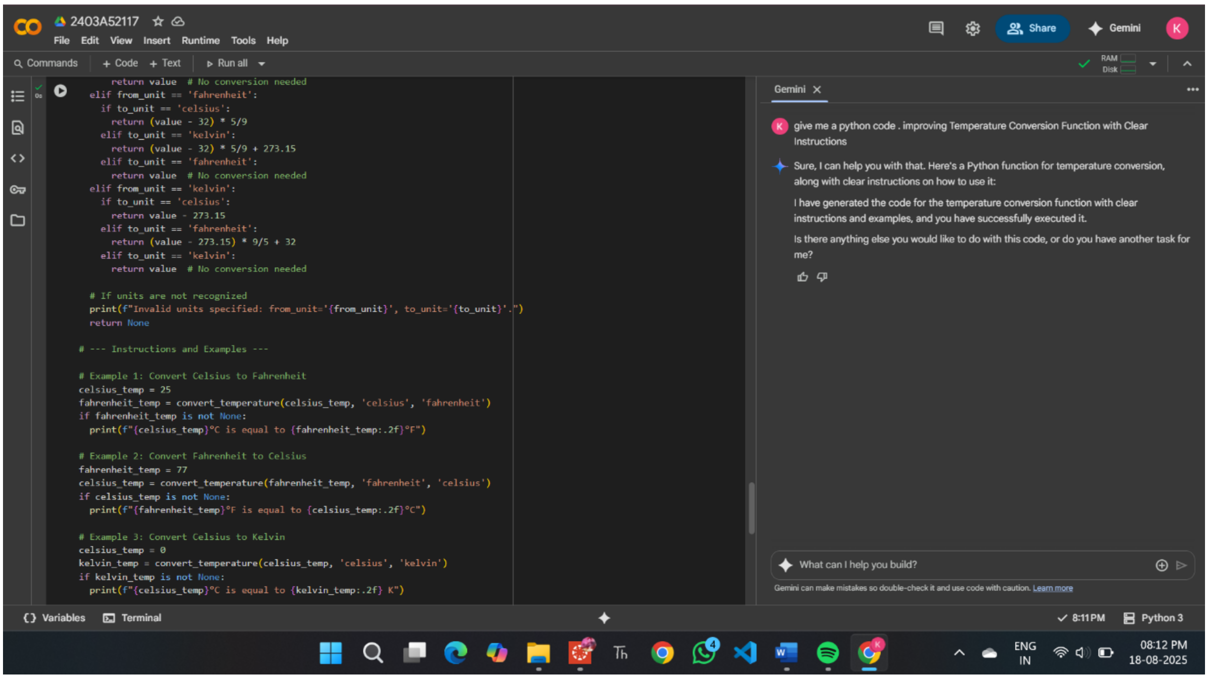
1. **Import necessary libraries:** It imports `tkinter` as `tk` and `ttk` (themed widgets).
2. **Define `HotelBillingApp` class:** This class encapsulates the application's logic and structure.
    - `__init__(self, root)`: This is the constructor. It takes a `root` argument, which is the main window of the application (`tk.Tk()`). It sets the window title and creates several labels and entry fields for the customer name, room type, and number of nights using `ttk.Label` and `ttk.Entry`. It also creates a "Generate Bill" button (`ttk.Button`) and a label to display the bill (`ttk.Label`). The `.grid()` method is used to arrange these widgets in a grid layout within the window.
    - `generate_bill(self)`: This method is intended to handle the bill generation logic when the button is clicked. Currently, it only prints "Generate Bill button clicked" and does nothing else (`pass`).
3. **Main execution block:**
    - `if __name__ == "__main__":`: This ensures the code inside this block only runs when the script is executed directly (not when imported as a module).
    - `root = tk.Tk()`: Creates the main application window.
    - `app = HotelBillingApp(root)`: Creates an instance of the `HotelBillingApp` class, passing the main window to it.
    - `root.mainloop()`: Starts the tkinter event loop, which listens for events (like button clicks) and keeps the window open until it's closed.

The error `TclError: no display name and no $DISPLAY environment variable` indicates that the code is trying to create a graphical user interface (GUI) in an environment where there is no display available, such as a server or a Colab notebook environment that doesn't have a direct graphical output set up. Tkinter applications typically need a graphical display to run.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

TASK 5:



```
      return value  # No conversion needed
  elif from_unit == 'fahrenheit':
    if to_unit == 'celsius':
      return (value - 32) * 5/9
    elif to_unit == 'kelvin':
      return (value - 32) * 5/9 + 273.15
    elif to_unit == 'fahrenheit':
      return value  # No conversion needed
  elif from_unit == 'kelvin':
    if to_unit == 'celsius':
      return value - 273.15
    elif to_unit == 'fahrenheit':
      return (value - 273.15) * 9/5 + 32
    elif to_unit == 'kelvin':
      return value  # No conversion needed

  # If units are not recognized
  print(f"Invalid units specified: from_unit='{from_unit}', to_unit='{to_unit}'.")
  return None

# --- Instructions and Examples ---

# Example 1: Convert Celsius to Fahrenheit
celsius_temp = 25
fahrenheit_temp = convert_temperature(celsius_temp, 'celsius', 'fahrenheit')
if fahrenheit_temp is not None:
  print(f"{celsius_temp}°C is equal to {fahrenheit_temp:.2f}°F")

# Example 2: Convert Fahrenheit to Celsius
fahrenheit_temp = 77
celsius_temp = convert_temperature(fahrenheit_temp, 'fahrenheit', 'celsius')
if celsius_temp is not None:
  print(f"{fahrenheit_temp}°F is equal to {celsius_temp:.2f}°C")

# Example 3: Convert Celsius to Kelvin
celsius_temp = 0
kelvin_temp = convert_temperature(celsius_temp, 'celsius', 'kelvin')
if kelvin_temp is not None:
  print(f"{celsius_temp}°C is equal to {kelvin_temp:.2f} K")
```

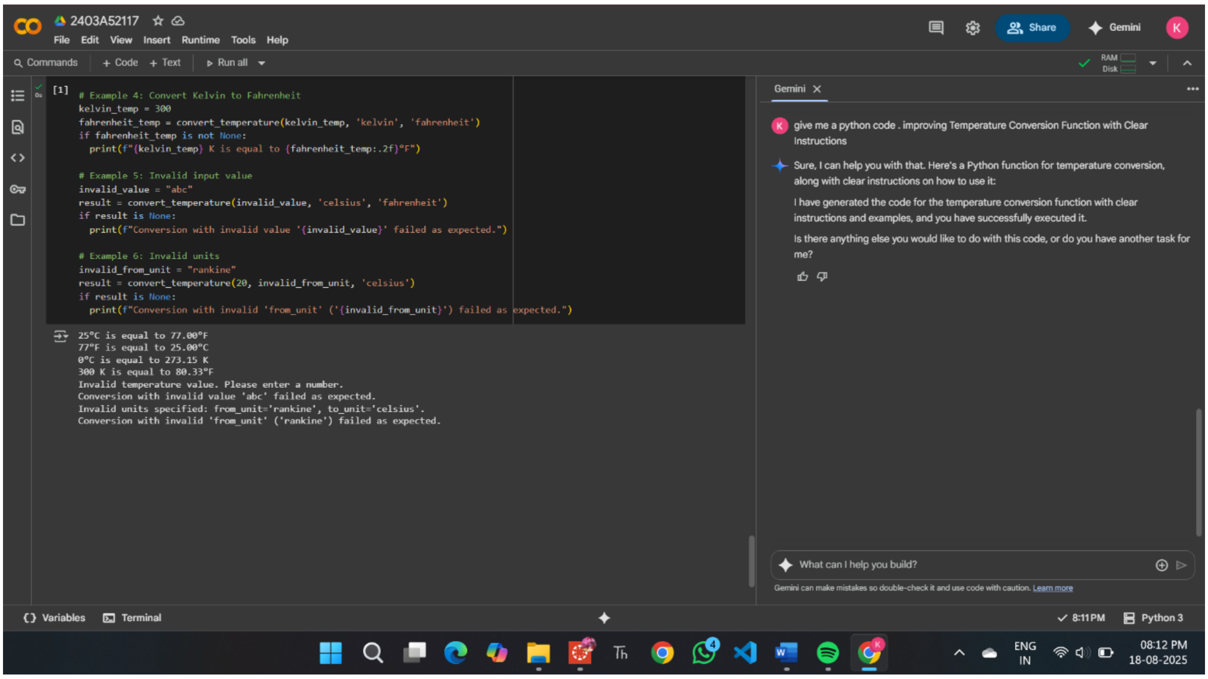
Gemini

K give me a python code . improving Temperature Conversion Function with Clear Instructions

Sure, I can help you with that. Here's a Python function for temperature conversion, along with clear instructions on how to use it:

I have generated the code for the temperature conversion function with clear instructions and examples, and you have successfully executed it.

Is there anything else you would like to do with this code, or do you have another task for me?



```
[1]
# Example 4: Convert Kelvin to Fahrenheit
kelvin_temp = 300
fahrenheit_temp = convert_temperature(kelvin_temp, 'kelvin', 'fahrenheit')
if fahrenheit_temp is not None:
  print(f"{kelvin_temp} K is equal to {fahrenheit_temp:.2f}°F")

# Example 5: Invalid input value
invalid_value = "abc"
result = convert_temperature(invalid_value, 'celsius', 'fahrenheit')
if result is None:
  print(f"Conversion with invalid value '{invalid_value}' failed as expected.")

# Example 6: Invalid units
invalid_from_unit = "rankine"
result = convert_temperature(20, invalid_from_unit, 'celsius')
if result is None:
  print(f"Conversion with invalid 'from_unit' ('{invalid_from_unit}') failed as expected.")
```

```
25°C is equal to 77.00°F
77°F is equal to 25.00°C
0°C is equal to 273.15 K
300 K is equal to 80.33°F
Invalid temperature value. Please enter a number.
Conversion with invalid value 'abc' failed as expected.
Invalid units specified: from_unit='rankine', to_unit='celsius'.
Conversion with invalid 'from_unit' ('rankine') failed as expected.
```

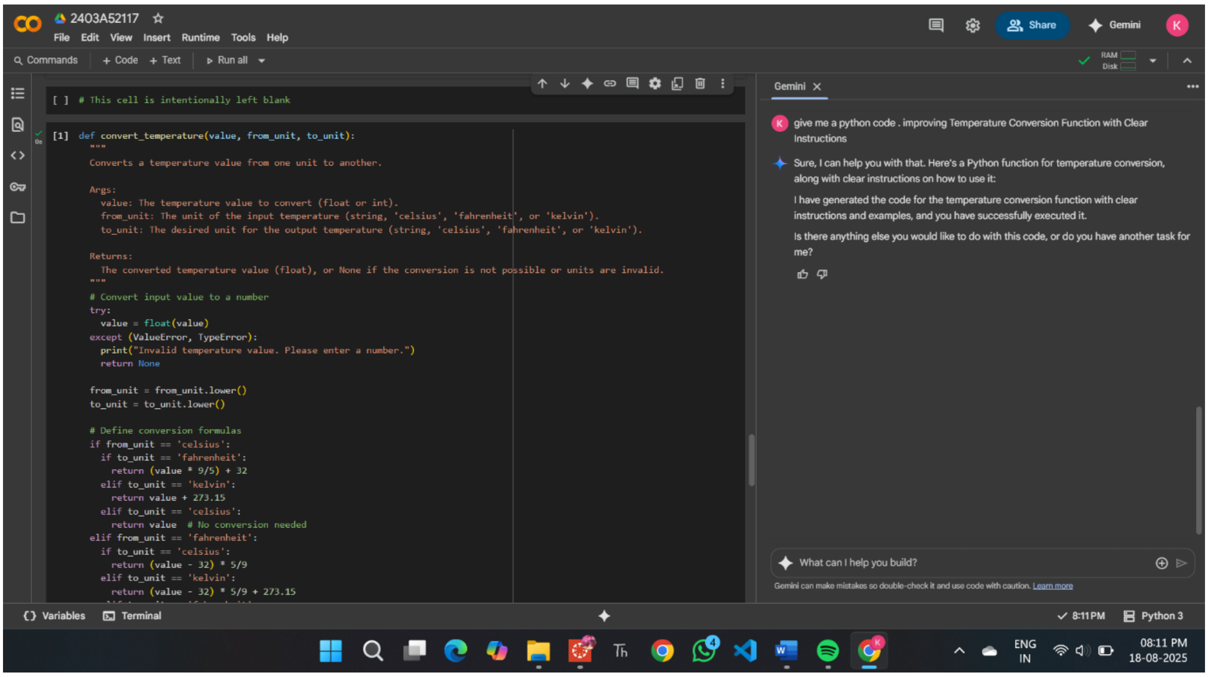
Gemini

K give me a python code . improving Temperature Conversion Function with Clear Instructions

Sure, I can help you with that. Here's a Python function for temperature conversion, along with clear instructions on how to use it:

I have generated the code for the temperature conversion function with clear instructions and examples, and you have successfully executed it.

Is there anything else you would like to do with this code, or do you have another task for me?



```
[ ] # This cell is intentionally left blank
```

```
[1] def convert_temperature(value, from_unit, to_unit):
      """
      Converts a temperature value from one unit to another.

      Args:
        value: The temperature value to convert (float or int).
        from_unit: The unit of the input temperature (string, 'celsius', 'fahrenheit', or 'kelvin').
        to_unit: The desired unit for the output temperature (string, 'celsius', 'fahrenheit', or 'kelvin').

      Returns:
        The converted temperature value (float), or None if the conversion is not possible or units are invalid.
      """
      # Convert input value to a number
      try:
        value = float(value)
      except (ValueError, TypeError):
        print("Invalid temperature value. Please enter a number.")
        return None

      from_unit = from_unit.lower()
      to_unit = to_unit.lower()

      # Define conversion formulas
      if from_unit == 'celsius':
        if to_unit == 'fahrenheit':
          return (value * 9/5) + 32
        elif to_unit == 'kelvin':
          return value + 273.15
        elif to_unit == 'celsius':
          return value  # No conversion needed
      elif from_unit == 'fahrenheit':
        if to_unit == 'celsius':
          return (value - 32) * 5/9
        elif to_unit == 'kelvin':
          return (value - 32) * 5/9 + 273.15
```

give me a python code . improving Temperature Conversion Function with Clear Instructions

Sure, I can help you with that. Here's a Python function for temperature conversion, along with clear instructions on how to use it:

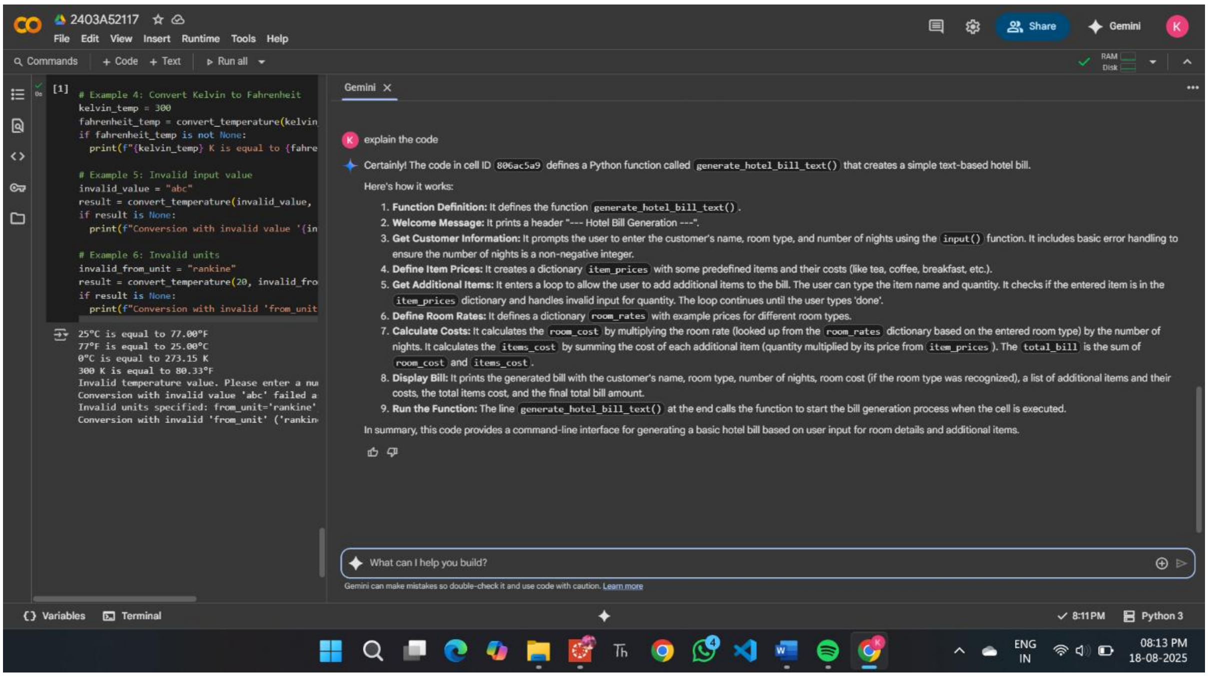
I have generated the code for the temperature conversion function with clear instructions and examples, and you have successfully executed it.

Is there anything else you would like to do with this code, or do you have another task for me?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

EXPLANATION:



```
[1] # Example 4: Convert Kelvin to Fahrenheit
    kelvin_temp = 300
    fahrenheit_temp = convert_temperature(kelvin
    if fahrenheit_temp is not None:
      print(f"{kelvin_temp} K is equal to {fahre

    # Example 5: Invalid input value
    invalid_value = "abc"
    result = convert_temperature(invalid_value,
    if result is None:
      print(f"Conversion with invalid value '{in

    # Example 6: Invalid units
    invalid_from_unit = "rankine"
    result = convert_temperature(20, invalid_fro
    if result is None:
      print(f"Conversion with invalid 'from_unit
```

```
25°C is equal to 77.00°F
77°F is equal to 25.00°C
0°C is equal to 273.15 K
300 K is equal to 80.33°F
Invalid temperature value. Please enter a nu
Conversion with invalid value 'abc' failed a
Invalid units specified: from_unit='rankine'
Conversion with invalid 'from_unit' ('rankin
```

explain the code

Certainly! The code in cell ID `806ac5a9` defines a Python function called `generate_hotel_bill_text()` that creates a simple text-based hotel bill.

Here's how it works:

1. **Function Definition:** It defines the function `generate_hotel_bill_text()`.
2. **Welcome Message:** It prints a header "--- Hotel Bill Generation ---".
3. **Get Customer Information:** It prompts the user to enter the customer's name, room type, and number of nights using the `input()` function. It includes basic error handling to ensure the number of nights is a non-negative integer.
4. **Define Item Prices:** It creates a dictionary `item_prices` with some predefined items and their costs (like tea, coffee, breakfast, etc.).
5. **Get Additional Items:** It enters a loop to allow the user to add additional items to the bill. The user can type the item name and quantity. It checks if the entered item is in the `item_prices` dictionary and handles invalid input for quantity. The loop continues until the user types 'done'.
6. **Define Room Rates:** It defines a dictionary `room_rates` with example prices for different room types.
7. **Calculate Costs:** It calculates the `room_cost` by multiplying the room rate (looked up from the `room_rates` dictionary based on the entered room type) by the number of nights. It calculates the `items_cost` by summing the cost of each additional item (quantity multiplied by its price from `item_prices`). The `total_bill` is the sum of `room_cost` and `items_cost`.
8. **Display Bill:** It prints the generated bill with the customer's name, room type, number of nights, room cost (if the room type was recognized), a list of additional items and their costs, the total items cost, and the final total bill amount.
9. **Run the Function:** The line `generate_hotel_bill_text()` at the end calls the function to start the bill generation process when the cell is executed.

In summary, this code provides a command-line interface for generating a basic hotel bill based on user input for room details and additional items.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more