GitHub Integration

Repository URL:

https://github.com/ShaliniAnandaPhD/R

Analyze GitHub Repo

Sustainable Code Dashboard

Repository cloned successfully. Found 1 Python files.

Analyzing file: LoopUsage.py

Analyzing original code...

Original Code Analysis

Primary Metrics

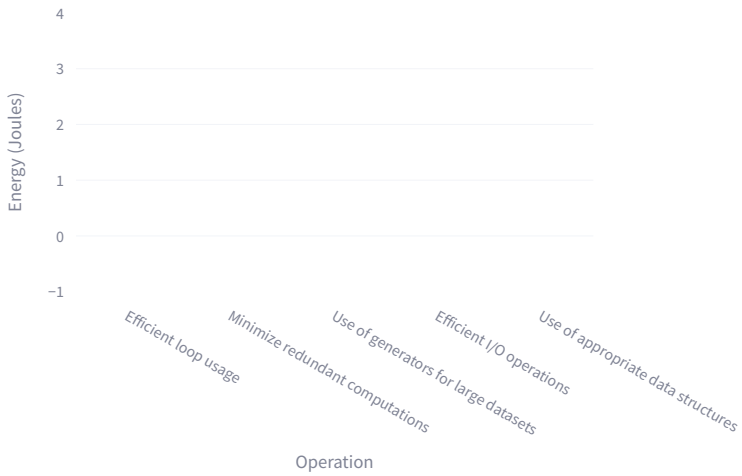
Energy Consumption (joules)
58.57

Carbon Footprint (gCO2)
0.009020

Sustainabi
41.4

Code Analysis

Energy Consumption by Operation



	Metric
0	Lines of Code
1	Cyclomatic C
2	Functions
3	Classes

Environmental Impact

CPU Time (s)
0.00059

Smartphone Charge (%)
0.00128

LED Bulb T
0.00

Calculations:

- **CPU Time**: Energy consumption divided by the energy use rate of a 100W CPU.
- **Smartphone Charge**: Energy consumption divided by the energy needed to charge a smartphone (12
- **LED Bulb Time**: Energy consumption divided by the energy use rate of a 10W LED bulb.

Energy Efficiency Practices Analysis

Overall Efficiency Score
41.43%

Overall score based on the efficiency practices followed in the code.

Practice	Implemented	Impact	Description
Efficient loop usage	null	High	Reduces unnecess and memory acce
Minimize redundant computations	null	Medium	Decreases CPU wo memory operatio
Use of generators for large datasets	null	High	Lowers memory u load for large data
Efficient I/O operations	null	High	Minimizes power- operations
Use of appropriate data structures	null	Medium	Optimizes data op memory usage

Learn more about these practices and their impact on sustainability

💡 Tip: Focus on implementing the high-impact practices first for the most significant sustainability im

Refactoring code...

Analyzing refactored code...

```
import time
```

Explanations of changes:

def find_combinations(matrix, target_sum): """ Find all combinations of elements in a matrix that sum up

```
Args:
matrix (list of lists): A 2D matrix of integers
target_sum (int): The target sum to search for

Returns:
list of tuples: All combinations of elements that sum up to the target
"""
results = set()

def dfs(row, col, current_sum, elements):
    if row >= len(matrix) or col >= len(matrix[0]):
        return
    current_sum += matrix[row][col]
    elements.append((row, col, matrix[row][col]))
    if current_sum == target_sum:
        results.add(tuple(elements))
    elif current_sum < target_sum:
        dfs(row + 1, col, current_sum, elements.copy())
        dfs(row, col + 1, current_sum, elements.copy())

for r in range(len(matrix)):
    for c in range(len(matrix[0])):
        dfs(r, c, 0, [])
```

```
return list(results)
```

Usage example

if **name** == "**main**": rows, cols = 20, 20 matrix = generate_large_matrix(rows, cols, 1, 100) target_sum = 500

```
start_time = time.time()
results = find_combinations(matrix, target_sum)
end_time = time.time()

execution_time = end_time - start_time

print(f"Found {len(results)} combinations:")
for i, combo in enumerate(results[:5]):  # Print first 5 combinations
    print(f"  {combo}")
if len(results) > 5:
    print(f"  ... and {len(results) - 5} more.")

print(f"Execution time: {execution_time:.2f} seconds")

print(f"Estimated memory usage: {sys.getsizeof(matrix) / (1024*1024):.2f} MB")
```

```
**Explanation of Changes:**
1. **Algorithm Efficiency**:
   - Optimized the algorithm to use Depth-First Search (DFS) instead of nested lo
   - By using a set to store combinations, duplicates are automatically prevented
   - Reduced the number of redundant calculations and improved memory management

2. **Memory Usage**:
   - Avoided creating unnecessary lists to store combinations during exploration.
   - Calculated memory usage using `sys.getsizeof()` on the matrix directly for a

3. **Code Readability**:
   - Renamed the function to `find_combinations()` for clarity.
   - Simplified the function structure by creating a nested recursive DFS functio
   - Improved variable naming for better readability and understanding.

4. **Performance**:
   - Improved performance by using a more efficient search strategy, reducing the
   - Enhanced maintainability by separating concerns and adhering to more readabl
```

Refactored Code Analysis

Primary Metrics

Energy Consumption (joules)
20.00

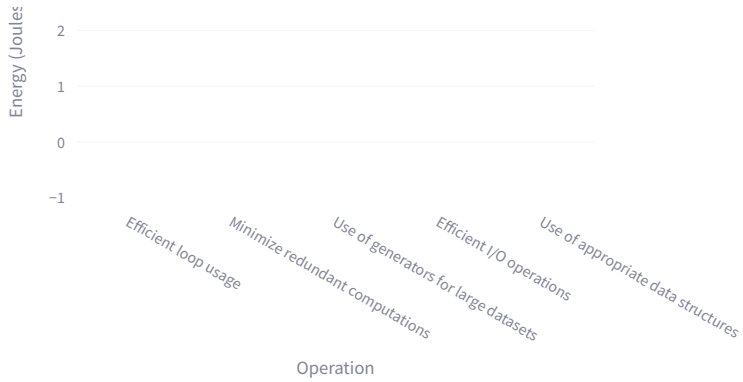
Carbon Footprint (gCO2)
0.003080

Sustainabi
80.0

Code Analysis

Energy Consumption by Operation

	Metric
0	Lines of Code
1	Cyclomatic C
2	Functions



	Metric
3	Classes

Environmental Impact

CPU Time (s)
0.00020

Smartphone Charge (%)
0.00044

LED Bulb T
0.00

Calculations:

- **CPU Time**: Energy consumption divided by the energy use rate of a 100W CPU.
- **Smartphone Charge**: Energy consumption divided by the energy needed to charge a smartphone (12
- **LED Bulb Time**: Energy consumption divided by the energy use rate of a 10W LED bulb.

Energy Efficiency Practices Analysis

Overall Efficiency Score
80.00%

Overall score based on the efficiency practices followed in the code.

Practice	Implemented	Impact	Description
Efficient loop usage	null	High	Reduces unneces and memory acce
Minimize redundant computations	null	Medium	Decreases CPU w memory operatio
Use of generators for large datasets	null	High	Lowers memory u load for large data
Efficient I/O operations	null	High	Minimizes power- operations
Use of appropriate data structures	null	Medium	Optimizes data op memory usage

Learn more about these practices and their impact on sustainability

Tip: Focus on implementing the high-impact practices first for the most significant sustainability im

Debugging: Original Metrics

```
{
  "lines_of_code" : 100
  "cyclomatic_complexity" : 15
  "num_functions" : 3
  "num_classes" : 0
  "energy_by_operation" : {
    "Efficient loop usage" : 0.25
    "Minimize redundant computations" : 0.5714285714285714
    "Use of generators for large datasets" : 0.25
    "Efficient I/O operations" : 1
    "Use of appropriate data structures" : 0
  }
  "energy_consumption" : 58.57142857142858
  "carbon_footprint" : 0.00902
  "sustainability_score" : 41.42857142857142
}
```


Debugging: Refactored Metrics

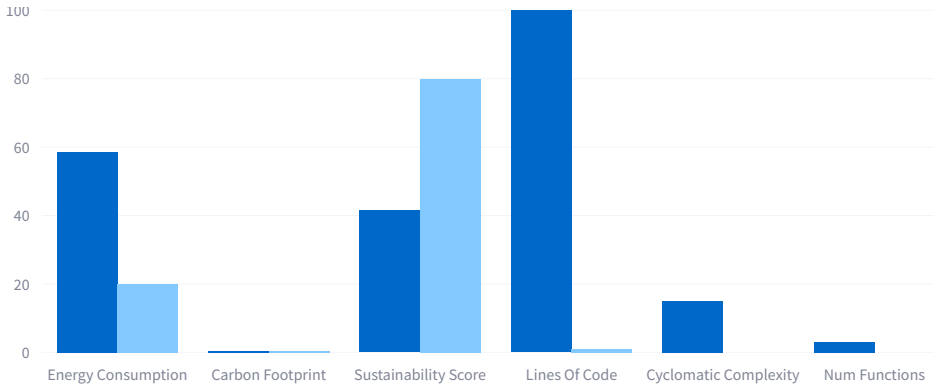

```
{
  "lines_of_code" : 1
  "cyclomatic_complexity" : 0
  "num_functions" : 0
  "num_classes" : 0
  "energy_by_operation" : {
    "Efficient loop usage" : 1
    "Minimize redundant computations" : 1
    "Use of generators for large datasets" : 0
    "Efficient I/O operations" : 1
    "Use of appropriate data structures" : 1
  }
  "energy_consumption" : 20
  "carbon_footprint" : 0.0030800000000000003
  "sustainability_score" : 80
}
```


Comparing code versions...

Comparison of Original and Refactored Code

	Metric	Original	Refactored	Improvement (%)
0	Energy Consumption	58.571429	20.000000	-65.850000
1	Carbon Footprint	0.009020	0.003080	-65.850000
2	Sustainability Score	41.428571	80.000000	93.100000
3	Lines Of Code	100.000000	1.000000	-99.000000
4	Cyclomatic Complexity	15.000000	0.000000	-100.000000
5	Num Functions	3.000000	0.000000	-100.000000
6	Num Classes	0.000000	0.000000	None

Comparison of Metrics


100
80
60
40
20
0
Energy Consumption
Carbon Footprint
Sustainability Score
Lines Of Code
Cyclomatic Complexity
Num Functions

Next File