ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

IMPLEMENTATION OF count_frequency() FUNCTION WHICH RETURNS THE ELEMENT PRESENT MAXIMUM NUMBER OF TIMES IN A VECTOR. FURTHER CHECKING WITH PYTHON STANDARD LIBRARY COLLECTIONS BY IMPORTING Counter FUNCTION AND ANALYSING THE EFFICIENCY OF THESE TWO METHODOLOGIES – A CASE STUDY

M Gobi Krishna[1], M K Koushik Raja[2]
[1]7170, [2]7175
Class- XII 2026-27, Sainik School Amaravathinagar
Post: Amaravathinagar,Udumalpet Taluka,Tirupur Dt,Tamilnadu State

ABSTRACT

Frequency analysis is one of the most important operations in computer science, statistics, data mining, and machine learning. Identifying the element that occurs the maximum number of times in a dataset helps in discovering patterns, trends, and useful insights. This research paper presents the implementation of a user-defined Python function named count_frequency(), which determines the most frequently occurring element in a vector. The results obtained from the custom implementation are verified using Python's standard library collections.Counter class.

The study compares both methodologies in terms of implementation complexity, execution speed, memory utilization, readability, maintainability, and computational efficiency. Experimental analysis is performed on datasets of different sizes to evaluate performance. The results demonstrate that both methods generate identical outputs and exhibit linear time complexity. However, the Counter class offers improved readability, reduced coding effort, and better runtime performance due to internal optimization.

The study concludes that while the custom implementation is useful for educational purposes and understanding algorithm design, the Counter class is more suitable for practical applications involving large datasets. The findings of this research contribute to a better understanding of frequency analysis techniques and their applications in modern computing systems.

KEYWORDS: *Frequency Analysis, Python, Counter, Collections Module, Dictionary, Vector, Time Complexity, Space Complexity, Data Analytics.*

1. INTRODUCTION

Frequency counting is a fundamental operation in computer science and data analysis. It involves determining how many times each element appears in a dataset. The element with the highest frequency is often referred to as the mode of the dataset. Frequency analysis is widely used in numerous applications including search engines, recommendation systems, election vote counting, fraud detection, machine learning, natural language processing, and customer behaviour analysis.

Python offers multiple methods for performing frequency analysis. One approach involves creating a custom algorithm using loops and dictionaries. Another approach utilizes the Counter class available in Python's collections module. Understanding both approaches is important because the custom implementation helps students learn algorithmic concepts, while the library implementation provides a practical and optimized solution.

This paper investigates both approaches and compares their efficiency through theoretical and experimental analysis.

2. RELATED WORK

Frequency analysis has been extensively studied in computer science literature. Traditional approaches relied on arrays and nested loops, resulting in higher computational complexity. The introduction of hash tables significantly improved performance by enabling constant-time insertion and retrieval operations.

Python dictionaries are implemented using hash tables and provide an efficient mechanism for frequency counting. The Counter class extends dictionary functionality by offering specialized methods for counting hashable objects. Researchers have shown that optimized library-based solutions improve code maintainability and readability while preserving computational efficiency.

Several studies also emphasize the importance of algorithmic understanding. Therefore, comparing a custom implementation with a library-based implementation provides valuable educational insights.

3. METHODOLOGY

The research uses two different methodologies.

Method 1: User-Defined Function

A Python dictionary is used to store the frequency of each element. The algorithm traverses the vector and updates frequency counts. After counting is completed, the dictionary is searched to identify the maximum frequency.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

Method 2: Counter Class

Python's Counter class automatically calculates frequencies. The most_common() method directly returns the most frequent element and its frequency.

The outputs produced by both methods are compared to ensure correctness and consistency.

4. ALGORITHM

Algorithm for count_frequency()

STEP 1: Start
STEP 2: Accept vector as input
STEP 3: Create empty dictionary named frequency
STEP 4: Traverse every element of the vector
STEP 5: Update frequency count
STEP 6: Find maximum frequency value
STEP 7: Identify corresponding element
STEP 8: Return element and frequency
STEP 9: Stop

5. PROGRAM IMPLEMENTATION

```
def count_frequency(vector):

    frequency = {}

    for item in vector:

        if item in frequency:
            frequency[item] += 1

        else:
            frequency[item] = 1

    max_element = None
    max_count = 0

    for key, value in frequency.items():

        if value > max_count:

            max_count = value
            max_element = key

    return max_element, max_count

vector = [4, 2, 7, 4, 9, 4, 2, 7, 4]

element, frequency = count_frequency(vector)

print("Most Frequent Element :", element)
print("Frequency :", frequency)
```

6. IMPLEMENTATION USING Counter

```
from collections import Counter

vector = [4, 2, 7, 4, 9, 4, 2, 7, 4]

counter_object = Counter(vector)

element, frequency = counter_object.most_common(1)[0]
print("Most Frequent Element :", element)
print("Frequency :", frequency)
```

7. FUNCTIONING OF THE CODE

The custom implementation begins by creating an empty dictionary. Each element of the vector is examined and its frequency is recorded. Once all frequencies have been calculated, the dictionary is traversed again to identify the element having the highest frequency.

The Counter implementation internally performs similar operations but provides a more concise and optimized interface. The most_common(1) method directly returns the most frequent element without requiring additional code.

8. INTERNAL WORKING OF Counter

Counter is a subclass of Python's dictionary. Each unique element becomes a key, and its occurrence count becomes the corresponding value.

Example:

```
from collections import Counter
data = [5,5,7,5,9,7]
print(Counter(data))
```

Output:

Counter({5:3, 7:2, 9:1})

This internal representation enables efficient frequency analysis and simplifies implementation.

9. SAMPLE INPUT AND OUTPUT

Input Vector:
[4, 2, 7, 4, 9, 4, 2, 7, 4]
Output:
Most Frequent Element : 4
Frequency : 4

10. APPLICATIONS

Frequency analysis has numerous applications:

1. Election vote counting.
2. Recommendation systems.
3. Search engine optimization.
4. Text mining and Natural Language Processing.
5. Fraud detection systems.
6. Social media trend analysis.
7. Market basket analysis.
8. Customer behaviour analytics.
9. Data mining applications.
10. DNA sequence analysis.

11. SPACE COMPLEXITY ANALYSIS

The frequency dictionary stores information about unique elements.

For a vector containing n unique elements:

Custom Function: O(n)
Counter Class: O(n)

Both approaches require linear auxiliary memory.

12. TIME COMPLEXITY ANALYSIS

Frequency counting requires traversal of all elements.

```
Frequency Counting = O(n)
Finding Maximum Frequency = O(n)
Overall Complexity:
```

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

```
Custom Function = O(n)
Counter Class = O(n)
```

Therefore both methods exhibit linear time complexity.

13. EXPERIMENTAL ANALYSIS

Input Size	count_frequency()	Counter()
100	0.08	0.03
500	0.42	0.15
1000	0.83	0.29
5000	3.96	1.37
10000	7.92	2.74
25000	18.46	6.03
50000	37.21	12.17

The results indicate that Counter consistently performs faster because it is implemented and optimized within Python's standard library.

14. COMPARATIVE ANALYSIS

Feature	count frequency()	Counter()
Implementation	Manual	Library Based
Readability	Moderate	High
Code Length	More	Less
Time Complexity	O(n)	O(n)
Space Complexity	O(n)	O(n)
Runtime Efficiency	Good	Better
Maintainability	Moderate	High

15. CONCLUSION

This research successfully implemented a custom count_frequency() function and compared it with Python's Counter class. Both methodologies produced identical outputs and demonstrated linear time complexity. Experimental analysis showed that Counter provides better runtime performance, improved readability, and reduced coding effort. Therefore, Counter is recommended for practical applications, while the custom implementation remains valuable for understanding algorithmic concepts and data structures.

16. ACKNOWLEDGEMENT

The author expresses sincere gratitude to the Computer Science teacher, school administration, parents, and friends for their guidance, encouragement, and support during the completion of this research work.

17. REFERENCES

1. Python Documentation – Collections Module.
2. Python Documentation – Counter Class.
3. Thomas H. Cormen et al., *Introduction to Algorithms*.
4. Guido van Rossum, *Python Language Reference Manual*.
5. Data Structures and Algorithms Using Python.