Code Similarity Analysis Report

Analysis Summary

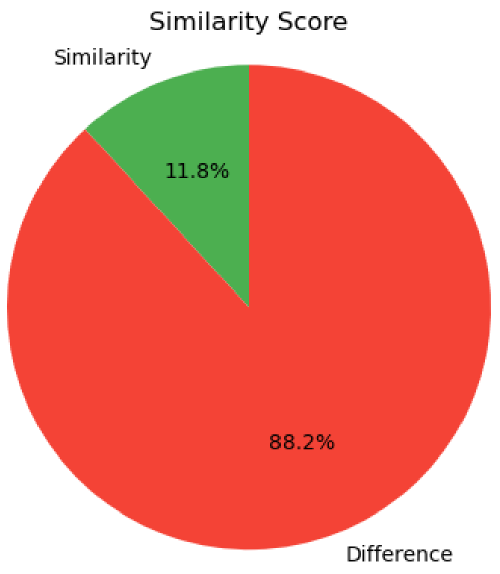
Comparison between: non_plagiarized1.py and non_plagiarized2.py

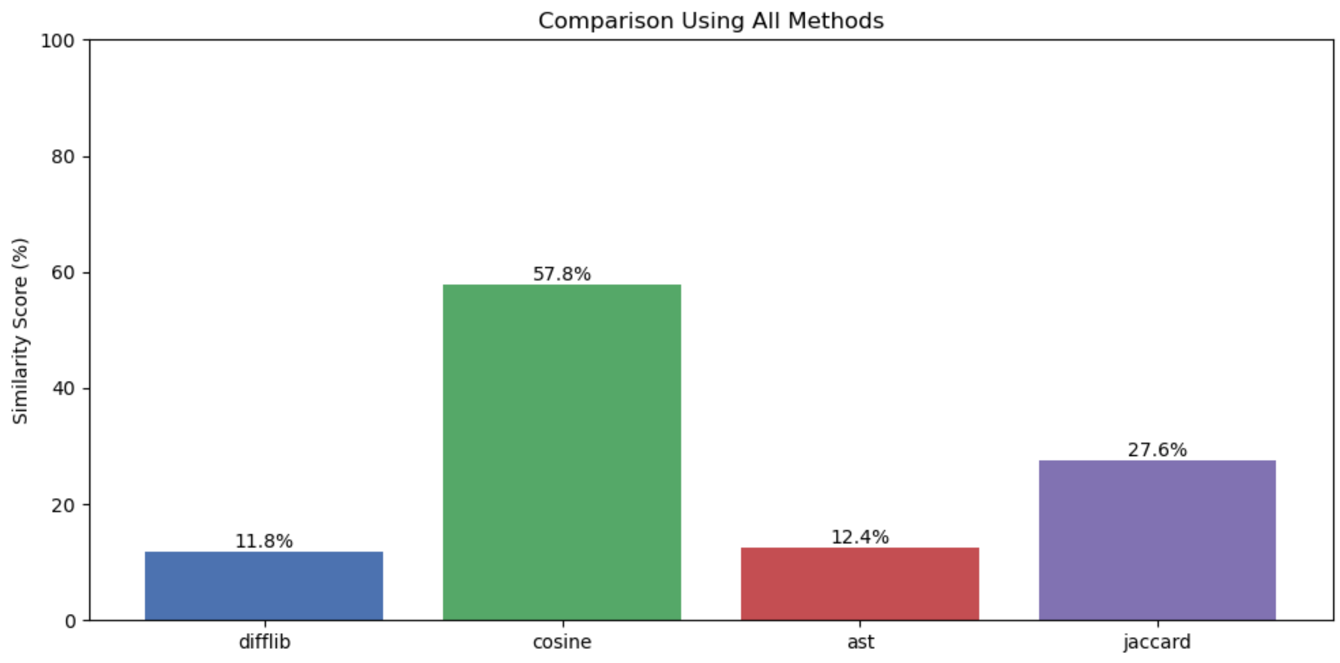
Selected Method: DIFFLIB

Similarity Score: 11.84%

Plagiarism Threshold (70%) Exceeded: No

Similarity Visualizations





Code Similarity Report

Preprocessing Details

Before comparison, the following preprocessing steps were applied:

1. All comments were removed

2. All identifiers were normalized (variables ? vN, functions ? fN, etc.)

Original vs Preprocessed Code

Original non_plagiarized1.py:

```
def sieve(limit):
    primes = [True] * (limit + 1)
    primes[0:2] = [False, False]
    for i in range(2, int(limit ** 0.5) + 1):
        if primes[i]:
            for j in range(i * i, limit + 1, i):
                primes[j] = False
    return primes

def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def next_k_sieve(start, k):
    upper = start + 1000
    sieve_list = sieve(upper)
    result = []
    for i in range(start + 1, upper):
        if sieve_list[i]:
            result.append(i)
            if len(result) == k:
                break
    return result

def main():
    num = int(input("Start number: "))
    k = int(input("Number of next primes to display: "))

    if is_prime_basic(num):
        print(f"{num} is prime.")
    else:
        print(f"{num} is not prime.")

    primes = next_k_sieve(num, k)
    print(f"Next {k} prime numbers are: {' '.join(map(str, ...
```

Code Similarity Report

Preprocessed non_plagiarized1.py:

```
def f0(p0):
    v0 = [True] * (limit + 1)
    primes[0:2] = [False, False]
    for v1 in range(2, int(limit ** 0.5) + 1):
        if primes[i]:
            for v2 in range(i * i, limit + 1, i):
                primes[j] = False
    return primes

def f1(p1):
    if n <= 1:
        return False
    for v1 in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def f2(p2, p3):
    v3 = start + 1000
    v4 = sieve(upper)
    v5 = []
    for v1 in range(start + 1, upper):
        if sieve_list[i]:
            result.append(i)
            if len(result) == k:
                break
    return result

def f3():
    v6 = int(input('Start number: '))
    v7 = int(input('Number of next primes to display: '))
    if is_prime_basic(num):
        print(f'{num} is prime.')
    else:
        print(f'{num} is not prime.')
    v0 = next_k_sieve(num, k)
    print(f"Next {k} prime numbers are: {' '.join(map(str, primes))}")
if __name__ == '__main__':
    main()
```

Original non_plagiarized2.py:

```
def is_prime_recursive(n, divisor=2):
    if n <= 2:
        return True if n == 2 else False
    if n % divisor == 0:
        return False
    if divisor * divisor > n:
        return True
    return is_prime_recursive(n, divisor + 1)
```

Code Similarity Report

```
def get_next_primes_recursive(start, count):
    result = []
    candidate = start + 1
    while len(result) < count:
        if is_prime_recursive(candidate):
            result.append(candidate)
        candidate += 1
    return result

def main():
    n = int(input("Enter number to check: "))
    total = int(input("How many primes to print after it: "))

    if is_prime_recursive(n):
        print(f"{n} is prime.")
    else:
        print(f"{n} is not prime.")

    print(f"Next {total} primes are:")
    print(" ".join(map(str, get_next_primes_recursive(n, total))))

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

Preprocessed non_plagiarized2.py:

```
def f0(p0, p1=2):
    if n <= 2:
        return True if n == 2 else False
    if n % divisor == 0:
        return False
    if divisor * divisor > n:
        return True
    return is_prime_recursive(n, divisor + 1)

def f1(p2, p3):
    v0 = []
    v1 = start + 1
    while len(result) < count:
        if is_prime_recursive(candidate):
            result.append(candidate)
        v1 += 1
    return result

def f2():
    v2 = int(input('Enter number to check: '))
    v3 = int(input('How many primes to print after it: '))
    if is_prime_recursive(n):
        print(f'{n} is prime.')
    else:
        print(f'{n} is not prime.')
```

Code Similarity Report

```
    print(f'Next {total} primes are:')
    print(' '.join(map(str, get_next_primes_recursive(n, total))))
if __name__ == '__main__':
    main()
```

Detailed Differences (Preprocessed Code)

```
--- file1
+++ file2
@@ -1,39 +1,29 @@
-def f0(p0):
-    v0 = [True] * (limit + 1)
-    primes[0:2] = [False, False]
-    for v1 in range(2, int(limit ** 0.5) + 1):
-        if primes[i]:
-            for v2 in range(i * i, limit + 1, i):
-                primes[j] = False
-    return primes
+def f0(p0, p1=2):
+    if n <= 2:
+        return True if n == 2 else False
+    if n % divisor == 0:
+        return False
+    if divisor * divisor > n:
+        return True
+    return is_prime_recursive(n, divisor + 1)

-def f1(p1):
-    if n <= 1:
-        return False
-    for v1 in range(2, int(n ** 0.5) + 1):
-        if n % i == 0:
-            return False
-    return True
-
-def f2(p2, p3):
-    v3 = start + 1000
-    v4 = sieve(upper)
-    v5 = []
-    for v1 in range(start + 1, upper):
-        if sieve_list[i]:
-            result.append(i)
-            if len(result) == k:
-                break
+def f1(p2, p3):
+    v0 = []
+    v1 = start + 1
+    while len(result) < count:
+        if is_prime_recursive(candidate):
+            result.append(candidate)
```

Code Similarity Report

```
+       v1 += 1
     return result

-def f3():
-    v6 = int(input('Start number: '))
-    v7 = int(input('Number of next primes to display: '))
-    if is_prime_basic(num):
-        print(f'{num} is prime.')
+def f2():
+    v2 = int(input('Enter number to check: '))
+    v3 = int(input('How many primes to print after it: '))
+    if is_prime_recursive(n):
+        print(f'{n} is prime.')
     else:
-        print(f'{num} is not prime.')
-    v0 = next_k_sieve(num, k)
-    print(f"Next {k} prime numbers are: {' '.join(map(str, primes))}")
+        print(f'{n} is not prime.')
+    print(f'Next {total} primes are:')
+    print(' '.join(map(str, get_next_primes_recursive(n, total))))
 if __name__ == '__main__':
     main()
```