assessment

September 25, 2024

0.1 Q.1 Optimized Prime Summation

0.1.1 Optimized Prime Summation Code:

[1]:
```
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def sum_primes_excluding_3(limit):
    prime_sum = 0
    for p in range(5, limit + 1):
        if is_prime(p) and p % 10 != 3:
            prime_sum += p
    return prime_sum

prime_sum = sum_primes_excluding_3(10_000_000)
print("Sum of primes from 5 to 10 million (excluding primes ending in 3):",␣
 ↪prime_sum)
```

```
Sum of primes from 5 to 10 million (excluding primes ending in 3): 2402006986464
```

1 Explanation:

1.1 The code has been optimized by incorporating the sum_primes_excluding_3 function, which computes the sum of primes within the specified range while excluding those ending in 3.Improved readability and maintainability through meaningful function and variable names.Usage of underscores in numeric literals for better readability.

2 Question: 2

2.1 1.Recursive Fibonacci Number Program:

[3]:
```
def fibonacci_recursive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

n = int(input("Enter the value of n: "))
print("The nth Fibonacci number using recursion is:", fibonacci_recursive(n))
```

```
Enter the value of n: 34
The nth Fibonacci number using recursion is: 5702887
```

2.2 2.Iterative Fibonacci Number Program:

[4]:
```
def fibonacci_iterative(n):
    if n <= 1:
        return n
    fib_prev, fib_curr = 0, 1
    for _ in range(2, n + 1):
        fib_prev, fib_curr = fib_curr, fib_prev + fib_curr
    return fib_curr

n = int(input("Enter the value of n: "))
print("The nth Fibonacci number using iteration is:", fibonacci_iterative(n))
```

```
Enter the value of n: 54
The nth Fibonacci number using iteration is: 86267571272
```

2.3 3.Time Complexity Comparison:

2.3.1 The recursive approach to finding Fibonacci numbers has exponential time complexity, specifically O(2^n), due to redundant calculations and repeated function calls.In contrast, the iterative approach has linear time complexity, O(n), as it computes each Fibonacci number once and stores the results to calculate subsequent numbers efficiently.

2.4 a.Palindrome Check Function:

[5]:
```python
def is_palindrome(s):
    return s == s[::-1]

string = input("Enter a string: ")
if is_palindrome(string):
    print("The entered string is a palindrome.")
else:
    print("The entered string is not a palindrome.")
```

```
Enter a string: 77
The entered string is a palindrome.
```

2.5 b.Count Palindromic Substrings Program:

[6]:
```python
def count_palindromic_substrings(s):
    count = 0
    n = len(s)
    for i in range(n):
        for j in range(i, n):
            if s[i:j + 1] == s[i:j + 1][::-1]:
                count += 1
    return count

string = input("Enter a string: ")
print("Number of palindromic substrings:", count_palindromic_substrings(string))
```

```
Enter a string: 88
Number of palindromic substrings: 3
```

2.6 Optimization Strategies:

2.6.1 Utilize dynamic programming to avoid redundant palindrome checks for substrings.Implement an algorithm that expands around the center of each potential palindrome to reduce time complexity to O(n^2).Explore efficient data structures such as suffix trees or suffix arrays to optimize substring searching for palindromes.

2.7 Q:4

[7]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
```

[8]:
```
iris_data = load_iris()
```

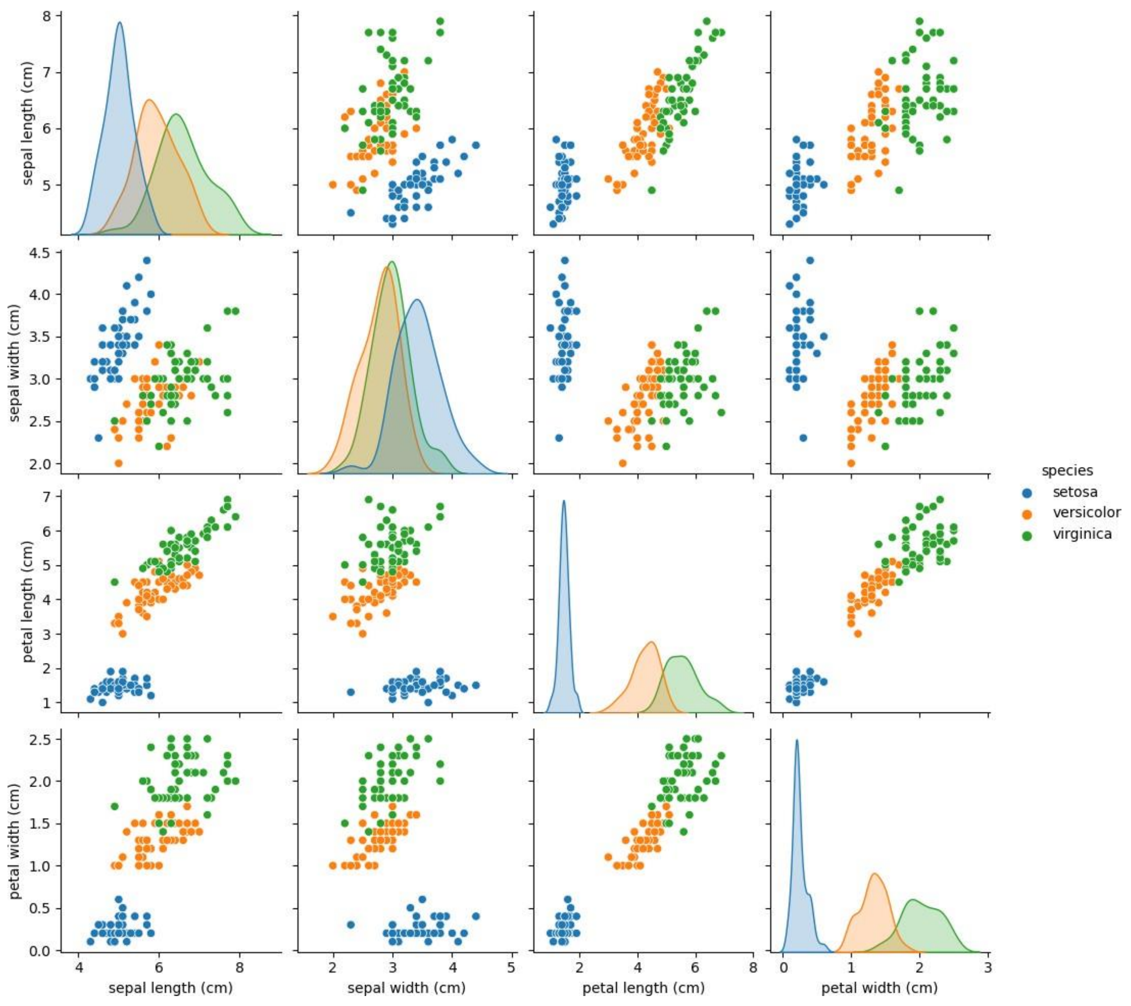
[9]:
```
iris_df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
iris_df['species'] = iris_data.target_names[iris_data.target]
```

[10]:
```
print("Dataset Information:")
print(iris_df.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    object
```

```
/home/sami/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

```
sns.pairplot(iris_df, hue='species')
plt.show()
```



[12]:
```
X = iris_df.drop('species', axis=1)
y = iris_df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[13]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support
```

```
      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Accuracy Score: 1.0

/home/sami/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

We use load_iris function from scikit-learn to load the Iris dataset directly. The data is then converted into a pandas DataFrame for easier manipulation and analysis. We explore the structure of the dataset using info() method to check for any missing values or data types. Visualizations using pair plots aid in understanding the relationships between different features and species. The dataset is split into training and testing sets using train_test_split function from scikit-learn. We train a Logistic Regression model on the training data and evaluate its performance using classification metrics.

```
[ ]:
```