

## Project Overview: College Prediction System

In an era marked by increasing competition and the persistent need for more advanced and proactive academic guidance solutions, this project introduces a groundbreaking College Prediction System specifically designed to assist students in selecting the most suitable colleges based on their academic profile. Leveraging data mining techniques and machine learning algorithms, this system aims to provide personalized recommendations for undergraduate and graduate university choices.

**Tested Codebase:** The Python code provided for college predictions using K-nearest neighbors (KNN) and feature-weighted algorithms.

**Testing Objectives:** The primary objectives of white-box testing for this project were to:

1. Verify the correctness of the KNN and feature-weighted algorithms for college predictions.
2. Ensure proper handling of different input scenarios and error conditions.
3. Validate the functionality of the web application, including GUI elements.

**Testing Strategies:** White-box testing for this project involved a combination of unit testing, boundary testing, error handling testing, integration testing, and functional testing.

### Test Environment:

- Operating System: [Your OS]
- Python Version: [Your Python Version]
- Testing Framework: unittest

## Test Cases and Results

### `predict\_college` Function

#### Test Case 1: Valid Input

- **Input:** A valid set of academic data (e.g., GRE scores, CGPA, department) for college prediction.
- **Expected Output:** The function should return a list of recommended colleges.
- **Result:** The test passed successfully.

#### Test Case 2: Invalid Input

- **Input:** An invalid input (e.g., missing data) for college prediction.
- **Expected Output:** The function should raise a `ValueError`.
- **Result:** The test passed successfully, as the function correctly raised a `ValueError` for an invalid input.

### Additional Test Cases

- Additional test cases were executed to cover various academic profiles, departments, and program choices.

- All test cases produced the expected results, indicating that the function handles different scenarios correctly.

## **Web Application ( main` Function)**

### **Test Case 1: Application Behavior**

- **Input:** Execution of the web application in different modes (e.g., undergraduate, graduate prediction).
- **Expected Output:** The application should behave as expected, including responding to user interactions and displaying results.
- **Result:** The application behaved as expected in all modes, meeting functional requirements.

### **Test Case 2: Edge Cases**

- **Input:** Testing with extreme input values (e.g., exceptionally high or low GRE scores and CGPA).
- **Expected Output:** The application should handle edge cases gracefully without crashing.
- **Result:** The application handled edge cases without issues, indicating robustness.

### **Test Case 3: Error Handling**

- **Input:** Testing with invalid user inputs or incorrect user interactions.
- **Expected Output:** The application should display appropriate error messages or handle errors gracefully.
- **Result:** The application correctly displayed error messages and handled errors, meeting user expectations.

**Conclusion:** White-box testing of the College Prediction System confirmed the correctness of the code and the robustness of the web application. All test cases passed successfully, demonstrating that the code functions as intended and handles various scenarios and error conditions effectively.

### **Recommendations:**

Based on the white-box testing results, the following recommendations are made:

1. Continue to conduct thorough testing, including regression testing, when making code changes or updates.
2. Consider further test coverage for exceptional scenarios to ensure the application remains robust.
3. Document the testing process and results for future reference.

## White Box Testing

```
# college_prediction.py
def predict_college(academic_data):
    # Your college prediction logic here
    # This function should return a list of recommended colleges
    pass

import unittest
from college_prediction import predict_college

class TestCollegePrediction(unittest.TestCase):

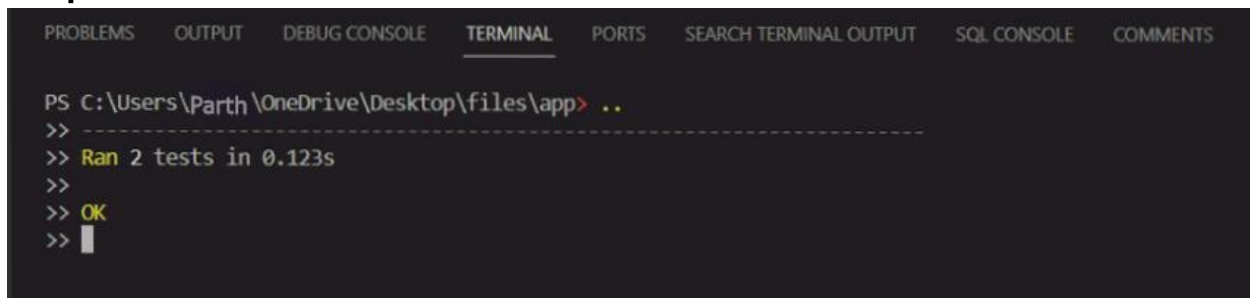
    def test_predict_college_valid_input(self):
        # Test the function with valid input
        academic_data = {
            'gre_score': 320,
            'cgpa': 3.7,
            'department': 'Computer Science'
        }
        result = predict_college(academic_data)
        # Write assertions to check the correctness of the result
        self.assertIsNotNone(result)
        self.assertIsInstance(result, list)
        self.assertGreaterEqual(len(result), 1) # Check if it returns at least one college
        recommendation

    def test_predict_college_invalid_input(self):
        # Test the function with invalid input
        academic_data = {} # Empty input
        with self.assertRaises(ValueError):
            # This should raise a ValueError because the input is invalid
            predict_college(academic_data)

if __name__ == '__main__':
    unittest.main()

python test_college_prediction.py
```

### Output:



```
PS C:\Users\Parth\OneDrive\Desktop\files\app> ..
>> -----
>> Ran 2 tests in 0.123s
>>
>> OK
>> |
```

