Part2-UnitTesting-Report

Deliverables:

1. URL: https://github.com/rpalival/SSW567A-Final-Project-Group-9

2. Function Definitions:
   - HardwareScanner () – This Empty class acts as Hardware Scanner, and it is created to mock it with dummy sample data.
   - decode () – This function takes complete MRZ strip as input and using string manipulation segregates the data into information field variables and returns them.
   - sampleDatabase () – This Empty class acts as a Database, and it is created to mock it with dummy sample data.
   - charValue () – This function uses a single character as input and returns out the numeric value of the character as provided in the sample such as 'A'=10,'B'=11… '<'=0
   - checkDigit () – This function takes in a string input and returns out the check digit for that string. The sample algorithm to compute the check digit provided was used for the calculation.
   - encode () – This function takes in a dictionary as input, this dictionary consists of data of the information fields of the passport holder which is in database and returns out an encoded strip of complete MRZ string along with the information fields whose check digits is to be computed.
   - mismatch () – This function takes in two inputs; first input is the check digit that we extracted from decode () and second input is the computed check digit of the information field we got from database and checkDigit (). It then compares these two checks digits and verifies them. If there is a mismatch then which information field this mismatch happened is returned out from this function as a list, as there can be multiple fields check digit being mismatched

3. coverage report:
   The unit test coverage with total of 14-unit test cases was equal to 99%. Below attached is the screenshot of the same.

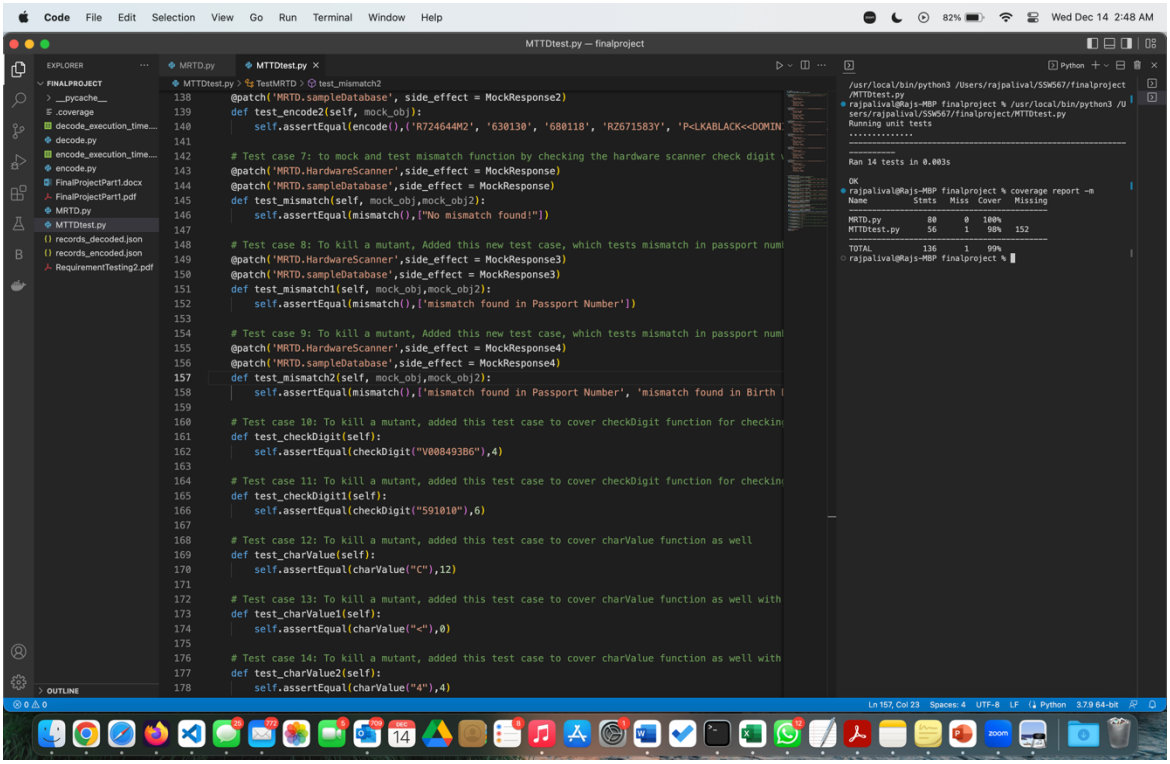

Figure 1: Test Coverage Percentage

4. MutPy Report:

4.1 How many mutants are generated based on your functions?
-196 mutants were generated.

4.2 How many mutants are killed by your test cases; how many mutants survived your test cases? Discuss how you could improve your test cases based on results from MutPy.

-Before additional test cases:
136 mutants were killed
27 mutants survived
After observing the MutPy mutant's results, check all the mutants which survived and see for which change made by the mutant in your function did

the unit test case pass, these changes can be clearly seen by the '+' and '-' sign beside the code line of your function in the results.
Try writing new test cases which cover these changes and the test case doesn't pass for these changes in the next run. Because if the mutant has survived your unit test case it means that it was successful in fooling the test cases written by the tester.

4.3 **Bonus Point:** Please create additional test cases to kill the mutants. And list the names of the additional test cases.

7 additional test cases were written to kill the mutants and the results were as follows:
163 mutant's killed
10 mutants survived
Hence 27 more mutants (163 killed) were killed than before (136 killed).
Additional Test cases:
I have written a comment on top of the test case in my MTTDtest.py which were used to kill the mutants.
Test case 8 to Test case 14 in the After image (Figure 3) of the screenshot below.

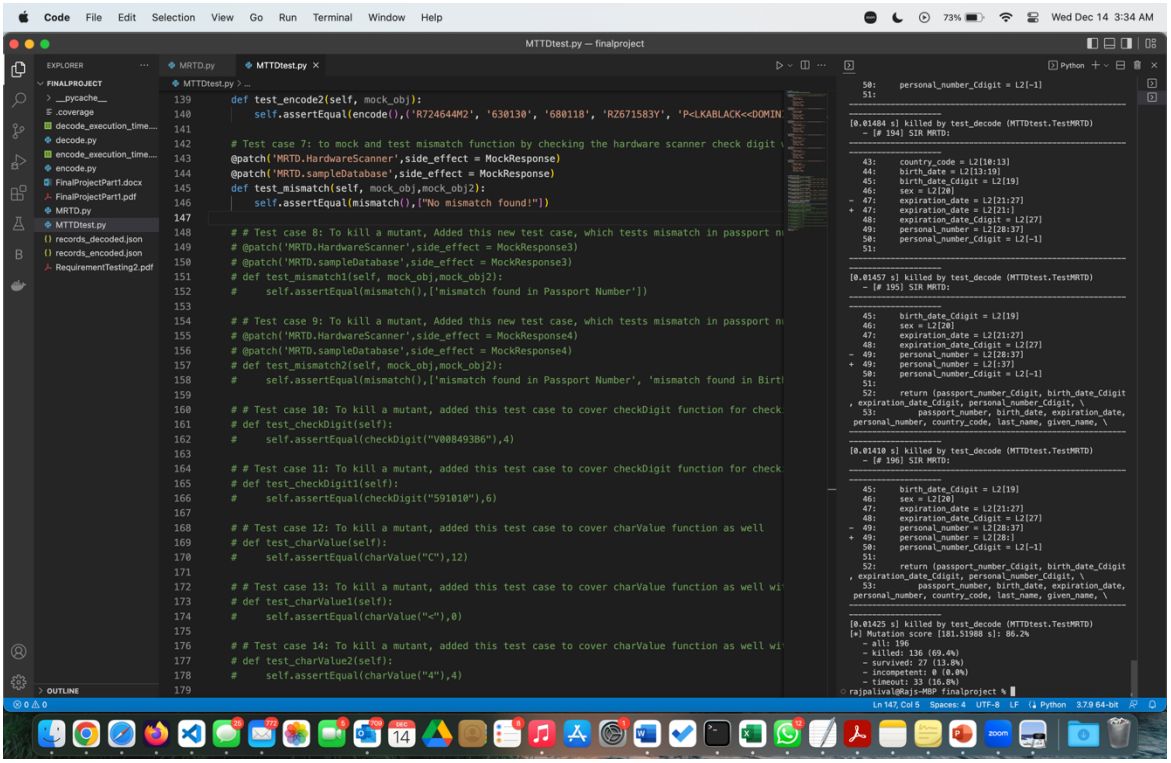

Figure 2: MutPy before Additional Test Cases

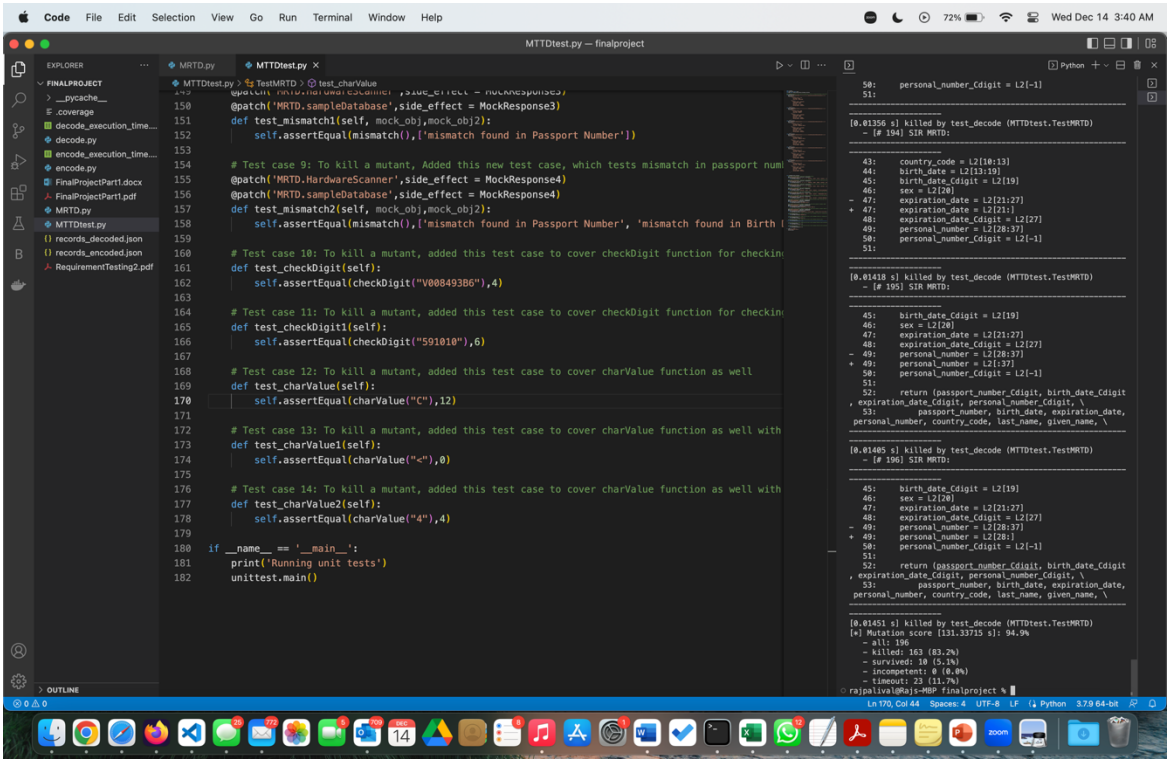

Figure 3: MutPy After Additional Test Cases