University of Prince Mugrin

College of Computer and Cyber Sciences

Department of Software Engineering

SE463- Software Testing and Quality Assurance

Course Project – Semester II (Spring 2023)

Testing the Sentiment Analysis Code

Team Members:

Jana Aldubai 4010372

Salwa Shama 4010405

Samah Shama 4010403

Sana Shama 4010404

Instructor:

Dr. Ftoon Kedwan

May 23, 2023

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 3

1.1 Project Description 3

1.2 Project Objective 3

CHAPTER 2: SOFTWARE TESTING 3

2.1 Software Testing Overview 3

2.2 Testing Principles 3

2.3 The Importance of Testing 4

2.4 Testing Types 4

2.5 Software Testing and Quality 4

CHAPTER 3: TESTING PLAN AND CRITERIA 4

3.1 Testing Plan 4

3.2 Coverage Criteria: Statement VS Branch VS Condition 5

CHAPTER 4: ANALYSIS AND DISCUSSION 6

4.1 Manual Testing 6

4.2 Automatic Testing Using Open-Source Testing Tool 12

4.4 Discussion of Manual Testing, and Automatic Testing Results 15

CHAPTER 5: CONCLUSION 16

5.1 Conclusion 16

5.2 References 16

APPENDICES 18

1. Members Roles 18

2. Sentiment Analysis Code 18

3. Coverage Report 19

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

CHAPTER 1: INTRODUCTION

1.1 Project Description

Our sentiment analysis software is a simple simulation for natural language processing technique that uses several if condition statements to automatically classify the sentiment of a given text into positive, negative, or neutral categories.

The program typically begins by pre-processing the text, which involves tasks such as tokenization to convert the raw text into a format that can be easily analysed. Then it uses a different set of terms to classify the tokens and calculate the percentages of positive words in the text, negative words in the text, and neutral words in the text. Finally, it provides us with the status of a person based on the high percentages that are computed.

1.2 Project Objective

The purpose of this project is to demonstrate our knowledge of Software Testing and Quality Assurance. To accomplish this, we will be utilizing dynamic testing as one of our testing techniques to measure code coverage. We will design our test plans, create test suites, and perform test coverage both manually and automatically. To compare the results, we will evaluate test suite coverage using the well-known "Coverage.py" library in Python. All of these concepts will be done side by side with improving our abilities to operate well as team players or leaders. Also, we will present the outcomes of the testing and quality assurance audit in a professional manner, both orally and in writing form.

CHAPTER 2: SOFTWARE TESTING

2.1 Software Testing Overview

Software development life cycle (SDLC) consists of four main stages which software testing is one of them. Software testing is the process of examine the software in different environments for different purposes before deploying it to the users. Software testing has many benefits. It includes decreasing defects, reducing costs, and enhances quality attributes such usability and performance [3].

2.2 Testing Principles

Software testing has seven principles. The first principle is that testing used to ensure the bugs' presence not absence. The second principle is that it is impossible to perform

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

fully comprehensive test for software. The third principle is the earlier performing testing and discover bugs the more you save cost and time. The fourth principle is that defects are not distributed equally. The fifth principle is that using the same tests every time will not let us discover new bugs. They always be changed and modified as needed. The sixth principle is that testing depend on the con the context or software type. The last principle is that it is fallacy to find free of bugs software [4].

2.3 The Importance of Testing

As software testing is one of the main stages in SDLC, it has high importance. The importance of quality control cannot be overstated when it comes to the development of software. Customers can become frustrated and lose faith in a brand if they receive a late delivery or have software defects. In worse scenarios, it is possible to have interconnected systems degraded or malfunction as the result of a bug or defect [3].

2.4 Testing Types

There are numerous types of software testing each with different benefits and approaches such as white box testing, black box testing, and non-functional testing. White box testing is testing the structure of the software including the code itself [5]. Black box testing is testing the product functions without seeing the code. It based on the system's requirements that are documented. The non-functional testing is testing characteristics of the system and quality attributes such as usability, reliability, and performance [6].

2.5 Software Testing and Quality

The relationship between software testing and quality assurance (QA) is that software testing discovers the bugs and defects that exist in the system while QA prevents them [7]. Software quality can be tested among different aspects called quality attributes such as performance, reliability, and usability. There are different testing strategies for different software models to ensure software quality. For example, in iterative and incremental models, regression testing is used [6].

CHAPTER 3: TESTING PLAN AND CRITERIA

3.1 Testing Plan

Software testing planning is key to ensuring that the testing effort, the spent time, and the consumed resources are well organized. It is also a super important piece of the puzzle for delivering high-quality software.

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

By the way, there is no silver bullet for testing because it tightly depends on the project itself. In our project "Sentiment Analysis", we will be following this plan:

Testing Plan	
Testing Type	Software Functionality Testing
Testing Strategy	Top-Down Approach
Testing Level	System Testing
Testing Technique	Dynamic Testing
Category of Testing Technique	White-Box Testing (Structure-Based)

Table 1 – Testing Plan

3.1.1 Test Cases

Test Item	Features to be Tested	Inputs	Expected Outputs	Pass/Fail Criteria
Test Case 01	a sentence containing only positive words	I absolutely love this new car! Driving it brings me so much joy and I am so excited to take it on a road trip this weekend.	Positive 😍 The percentage of positive words in the text is: 11.11111111111111 % The percentage of negative words in the text is: 0.0 % The percentage of neutral words in the text is: 0.0 %	Get the expected value.
Test Case 02	a sentence containing only negative words	I'm so frustrated with this new phone. It's slow, the battery life is terrible, and it keeps freezing.	Negative 😢 Here's a joke for you to feel better: Why do programmers prefer dark mode? Because light attracts bugs. The percentage of positive words in the text is: 0.0 % The percentage of negative words in the text is: 5.555555555555555 % The percentage of neutral words in the text is: 0.0 %	Get the expected value.
Test Case 03	a sentence containing only neutral words	The weather today is normal and the traffic is average	Neutral 😐 The percentage of positive words in the text is: 0.0 % The percentage of negative words in the text is: 0.0 % The percentage of neutral words in the text is: 20.0 %	Get the expected value.
Test Case 04	a sentence containing a mix of positive, negative, and neutral words	I always feel a sense of joy and love when I spend time with my family, even if we're just doing something normal and average like watching TV. However, when someone in the family is angry or frustrated, it can quickly turn a happy moment into a sad and	The percentage of positive words in the text is: 5.88235294117647 % The percentage of negative words in the text is: 5.88235294117647 % The percentage of neutral words in the text is: 3.9215686274509802 %	Get the expected value.
Test Case 05	a sentence containing a mix of positive, and neutral words	I am so excited we are going to the zoo with my friend. I love monkeys and I am happy to see them in reality. However, she thinks it is a normal thing.	Positive 😘 The percentage of positive words in the text is: 9.090909090909092 % The percentage of negative words in the text is: 0.0 % The percentage of neutral words in the text is: 3.0303030303030303 %	Get the expected value.
Test Case 06	a sentence containing a mix of negative, and neutral words	I am so sad as my brother has an accident. It is an average accident and he is fine now. However, I hate seeing him in such a situation like this.	The percentage of positive words in the text is: 0.0 % The percentage of negative words in the text is: 6.451612903225806 % The percentage of neutral words in the text is: 6.451612903225806 %	Get the expected value.

Table 2 – Test Cases

3.2 Coverage Criteria: Statement VS Branch VS Condition

The goal of software testing is to deliver high-quality software, and that can be done by using many techniques. In this project, we will use structure-based techniques, but to be more precise, we will use from a toolkit of structure-based techniques, specifically statements, branch and condition coverage.

Statement Coverage: It measures the percentage of executable statements in a program that have been executed during the test at least once. [1]

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statements}} * 100$$

Branch Coverage: It measures the percentage of the possible paths in a program that have been executed during testing at least once.[1]

$$\text{Branch Coverage} = \frac{\text{Number of Executed Branches}}{\text{Total Number of Branches}} * 100$$

Condition Coverage: It measures the percentage of the executable condition in a program that have been executed during testing at least once.[1]

$$\text{Condition Coverage} = \frac{\text{Number of Executed Operands}}{\text{Total Number of Operands}} * 100$$

CHAPTER 4: ANALYSIS AND DISCUSSION

4.1 Manual Testing

Test Case 01:

Input	Output
I absolutely love this new car! Driving it brings me so much joy and I am so excited to take it on a road trip this weekend.	Positive 😍 The percentage of positive words in the text is: 11.11111111111111% The percentage of negative words in the text is: 0.0% The percentage of neutral words in the text is: 0.0 %

Table 1 – Test Case 01

For measuring the statement coverage manually, we need to know two things: we need to know the total number of LOC we have in our program, which in this case is 37 LOC, and the total number of statements that have been executed by this test case, which are [1, 3, from 5 to 7, from 9 to 11, 13, from 15 to 18, 20, from 24 to 26, from 28 to 30, 42 , from 44 to 46 and from 48 to 50]* , so the total is 27 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{27}{37} * 100 = 72.9\%$$

For measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [15, 16, 18, 20, 28, 31, and 38], but every decision has two branches [true, false], so the number becomes 14 branches. In terms of executed branches in this case, line 15 [true, false], line 16 [true, false], line 18 [- , false], line 20 [- , false], line 28 [- , true], line 31 [- , -], and line 38 [- , -], so it was executed 7 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{7}{14} * 100 = 50\%$$

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

For measuring the condition coverage manually, what is necessary to have them to calculate are also two things, the total number of operands that we have in the code, depending on the following lines we have [one in line 15, one in line 16, one in line 18, one in line 20, two in line 28, two in line 31 and two in line 38] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 15, [true, false] in line 16, [- , false] in 18, [- , false] in 20, [true , -][true , -] in line 28, [- , -][- , -] in line 31, [- , -][- , -] in line 38], so we have 8 conditions were executed, so let's do the math by the following equation and get the results:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{8}{20} * 100 = 40\%$$

Test Case 02:

Input	Output
I'm so frustrated with this new phone. It's slow, the battery life is terrible, and it keeps freezing.	Negative 😥 Here's a joke for you to feel better: Why do programmers prefer dark mode? Because light attracts bugs. The percentage of positive words in the text is: 0.0% The percentage of negative words in the text is: 5.555555555555555% The percentage of neutral words in the text is: 0.0 %

Table 2 – Test Case 02

As we said early for measuring the statement coverage manually, we need to know the total number of LOC we have in our program, which in this case is 37 LOC as we mentioned previously , and the total number of statements that have been executed by this test case, which are [1, 3, from 5 to 7, from 9 to 11, 13 , 15, 16 from 18 to 20, from 24 to 26, 28, from 31 to 34, 37, 42, from 44 to 46, and from 48 to 50], so the total is 30 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{30}{37} * 100 = 81.08\%$$

As we said early for measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [15, 16, 18, 20, 28, 31, and 38], but every decision has two branches [true, false], so the number becomes 14 branches as we mentioned previously. In terms of executed branches in this case, line 15 [true, false], line 16 [- , false], line 18 [false, true], line 20 [- , false], line 28 [- , false], line 31 [- , true], and line 38 [- , -], so it was executed 8 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{8}{14} * 100 = 57.14\ \%$$

For measuring the condition coverage manually, as we mentioned early we have to get also two things, first the total number of operands that we have in the code, depending on the following lines we have [one in line 15, one in line 16, one in line 18, one in line 20, two in line 28, two in line 31 and two in line

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

38] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 15, [-, false] in line 16, [true, false] in 18, [- , false] in 20, [- , false][- , -] in line 28, [true , -][true , -] in line 31, [- , -][- , -] in line 38], so we have 10 conditions were executed, so let's do the math by the following equation and get the results as we did previously:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{9}{20} * 100 = 45\ \%$$

Test Case 03:

Input	Output
The weather today is normal and the traffic is average	Neutral 😐 The percentage of positive words in the text is: 0.0% The percentage of negative words in the text is: 0.0% The percentage of neutral words in the text is: 20.0 %

Table 3 – Test Case 03

As we said early for measuring the statement coverage manually for test case 03, we need to know the total number of LOC we have in our program, which in this case is 37 LOC as we mentioned previously, and the total number of statements that have been executed by this test case, which are [1, 3, from 5 to 7, from 9 to 11, 13, 15, 16, 18, 20, 21, from 24 to 26, 28, 31, 38, 39, 40, 42, from 44 to 46, from 48 to 50], so the total is 29 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{29}{37} * 100 = 78.37\%$$

As we said early for measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [15, 16, 18, 20, 28, 31, and 38], but every decision has two branches [true, false], so the number becomes 14 branches as we mentioned previously. In terms of executed branches in this case, line 15 [true, false], line 16 [false], line 18 [false], line 20 [false, true], line 28 [false], line 31 [false], and line 38 [true], so it was executed 8 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{9}{14} * 100 = 64.28\%$$

For measuring the condition coverage manually, as we mentioned early we have to get also two things, first the total number of operands that we have in the code, depending on the following lines we have [one in line 15, one in line 16, one in line 18, one in line 20, two in line 28, two in line 31 and two in line 38] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 15, [false] in line 16, [false] in 18, [false, true] in 20, [false] in line 28, [false] in line 31, [true][true] in line 38], so we

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

have 12 conditions were executed, so let's do the math by the following equation and get the results as we did previously:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{10}{20} * 100 = 50\%\ \%$$

Test Case 04:

Input	Output
I always feel a sense of joy and love when I spend time with my family, even if we're just doing something normal and average like watching TV. However, when someone in the family is angry or frustrated it can quickly turn a happy moment into a sad and tense one.	The percentage of positive words in the text is: 5.88235294117647% The percentage of negative words in the text is: 5.88235294117647% The percentage of neutral words in the text is: 3.9215686274509802 %

Table 4 – Test Case 04

For measuring the statement coverage manually, we need to know two things: we need to know the total number of LOC we have in our program, which in this case is 37 LOC, and the total number of statements that have been executed by this test case, which are [from 1 to 19, 20, 23, 28, and from 31 to 37], so the total is 29 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{29}{37} * 100 = 78.37\%$$

As we said early for measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [10, 11, 13, 15, 20, 23, and 28], but every decision has two branches [true, false], so the number becomes 14 branches as we mentioned previously. In terms of executed branches in this case, line 10 [true, false], line 11 [true, false], line 13 [true, false], line 15 [true, false], line 20 [false], line 23 [false], and line 28 [false], so it was executed 11 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{11}{14} * 100 = 78.57\%$$

For measuring the condition coverage manually, as we mentioned early we have to get also two things, first the total number of operands that we have in the code, depending on the following lines we have [one in line 10, one in line 11, one in line 13, one in line 15, two in line 20, two in line 23 and two in line 28] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 10, [true, false] in line 11, [true, false] in 13, [true, false] in 15, [false] in line 20, [false] in line 23, [false] in line 28], so we have 14 conditions were executed, so let's do the math by the following equation and get the results as we did previously:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{11}{20} * 100 = 75\%$$

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 05:

Input	Output
I am so excited we are going to the zoo with my friend. I love monkeys and I am happy to see them in reality. However, she thinks it is a normal thing.	Positive 🥰 The percentage of positive words in the text is: 9.090909091 % The percentage of negative words in the text is: 0.0 % The percentage of neutral words in the text is: 3.03030303 %

Table 5 – Test Case 05

Again for measuring the statement coverage manually, we need to know two things: we need to know the total number of LOC we have in our program, which in this case is 37 LOC, and the total number of statements that have been executed by this test case, which are [1, 4, 6-8, 10-12, 14, 16-19, 21-22, 25-27, 29-31, 43, 45-47, 49-51], so the total is 28 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{28}{37} * 100 = 75.67\%$$

Again for measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [16, 17, 19, 21, 29, 32, and 39], but every decision has two branches [true, false], so the number becomes 14 branches. In terms of executed branches in this case, line 16 [true, false], line 17 [true, false], line 19 [false], line 21 [true, false], line 29 [true], line 32 [-], and line 39 [-], so it was executed 8 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{8}{14} * 100 = 57.14\%$$

Again for measuring the condition coverage manually, as we mentioned early we have to get also two things, first the total number of operands that we have in the code, depending on the following lines we have [one in line 16, one in line 17, one in line 19, one in line 21, two in line 29, two in line 32 and two in line 39] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 16, [true, false] in line 17, [false] in 19, [true, false] in 21, [true][true] in line 29, [-][-] in line 32, [-][-] in line 39], so we have 13 conditions were executed, so let's do the math by the following equation and get the results as we did previously:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{9}{20} * 100 = 45\%$$

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 06:

Input	Output
I am so sad as my brother has an accident. It is an average accident and he is fine now. However, I hate seeing him in such a situation like this.	The percentage of positive words in the text is: 0.0% The percentage of negative words in the text is: 6.451612903225806% The percentage of neutral words in the text is: 6.451612903225806 %

Table 6 – Test Case 06

Again for measuring the statement coverage manually, we need to know two things: we need to know the total number of LOC we have in our program, which in this case is 37 LOC, and the total number of statements that have been executed by this test case, which are [1, 4, 6-8, 10-12, 14, 16-17, 19-22, 25-27, 29, 32, 39, 43, 45-47, 49-51], so the total is 28 lines of executed. Then we substitute them in the following formula and get the result:

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total Number of Statement}} * 100 = \frac{28}{37} * 100 = 75.67\%$$

Again for measuring the branch coverage manually, we also need to know two things: the first is the total number of branches that we have in this program, which is 7 in the following lines [16, 17, 19, 21, 29, 32, and 39], but every decision has two branches [true, false], so the number becomes 14 branches. In terms of executed branches in this case, line 16 [true, false], line 17 [false], line 19 [true, false], line 21 [true, false], line 29 [false], line 32 [false], and line 39 [false], so it was executed 10 branches, so we can apply the branch formula as follows:

$$\text{Branch Coverage} = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches} * 100 = \frac{10}{14} * 100 = 71.42\%$$

Again for measuring the condition coverage manually, as we mentioned early we have to get also two things, first the total number of operands that we have in the code, depending on the following lines we have [one in line 16, one in line 17, one in line 19, one in line 21, two in line 29, two in line 32 and two in line 39] we have 10, but as each condition has two outputs, the number becomes 20, the second thing we need it is that the number of executed operands by this test case, which are [[true, false] in line 16, [false] in line 17, [true, false] in 19, [true, false] in 21, [false][-] in line 29, [false][-] in line 32, [false][-] in line 39], so we have 13 conditions were executed, so let's do the math by the following equation and get the results as we did previously:

$$\text{Condition Coverage} = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands} * 100 = \frac{10}{20} * 100 = 50\%$$

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

4.2 Automatic Testing Using Open-Source Testing Tool

Test Case 01:

a. Statement Coverage Report for Test Case 01:

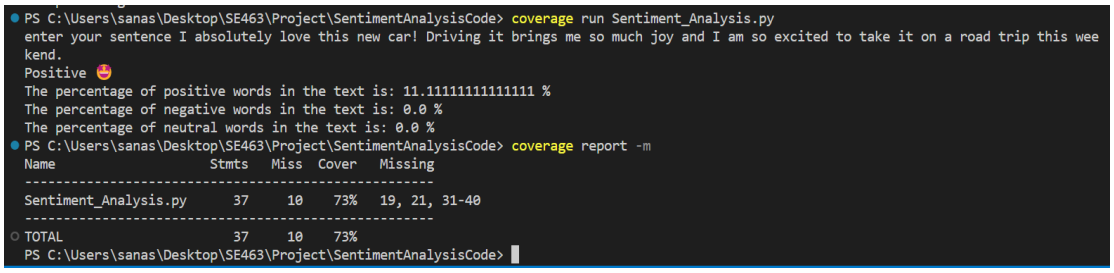

Figure 1 – Tool Output for Statement Test for Test Case 01

b. Branch Coverage Report for Test Case 01:

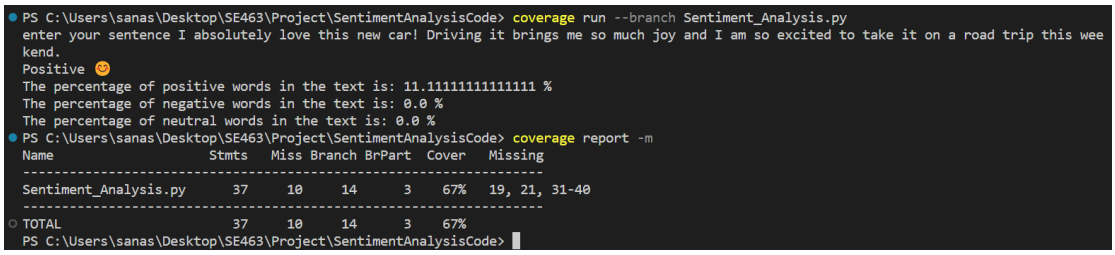

Figure 2 – Tool Output for Branch Test for Test Case 01

Test Case 02:

a. Statement Coverage Report for Test Case 02:

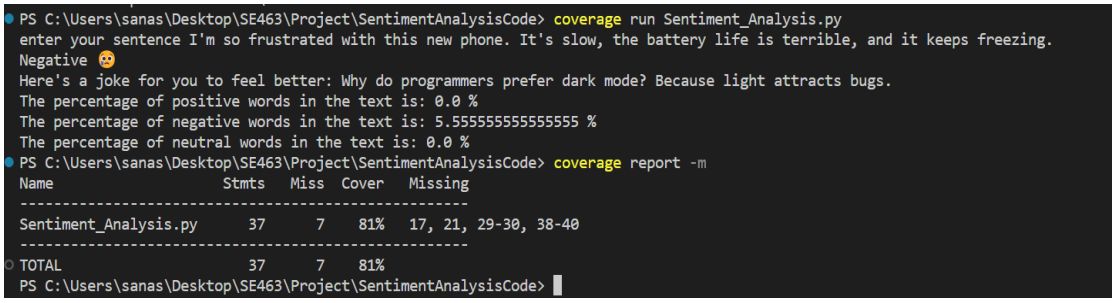

Figure 1 – Tool Output for Statement Test for Test Case 02

b. Branch Coverage Report for Test Case 02:

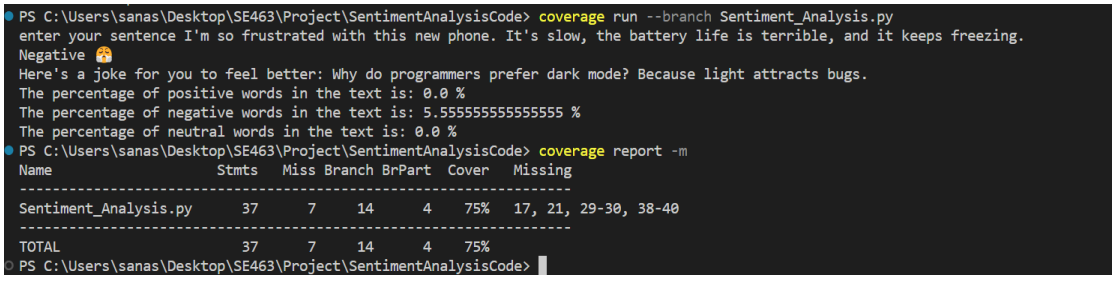

Figure 2 – Tool Output for Branch Test for Test Case 02

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 03:

a. Statement Coverage Report for Test Case 03:

```
PS C:\Users\lenovo\Desktop\SE463- Testing\PROJECT> coverage run SentimentAnalysis.py
Enter your sentence The weather today is normal and the traffic is average
Neutral 😐
The percentage of positive words in the text is: 0.0 %
The percentage of negative words in the text is: 0.0 %
The percentage of neutral words in the text is: 20.0 %
PS C:\Users\lenovo\Desktop\SE463- Testing\PROJECT> coverage report
Name                   Stmts   Miss  Cover
------------------------------------------
SentimentAnalysis.py      37      8    78%
------------------------------------------
TOTAL                     37      8    78%
```

Figure 1 – Tool Output for Statement Test for Test Case 03

b. Branch Coverage Report for Test Case 03:

```
PS C:\Users\lenovo\Desktop\SE463- Testing\PROJECT> coverage run --branch SentimentAnalysis.py
Enter your sentence The weather today is normal and the traffic is average
Neutral 😐
The percentage of positive words in the text is: 0.0 %
The percentage of negative words in the text is: 0.0 %
The percentage of neutral words in the text is: 20.0 %
PS C:\Users\lenovo\Desktop\SE463- Testing\PROJECT> coverage report
Name                   Stmts   Miss Branch BrPart  Cover
------------------------------------------------------
SentimentAnalysis.py      37      8     14      5    75%
------------------------------------------------------
TOTAL                     37      8     14      5    75%
```

Figure 2 – Tool Output for Branch Test for Test Case 03

Test Case 04:

a. Statement Coverage Report for Test Case 04:

```
C:\Users\samah\OneDrive\Desktop\SE463_Project>coverage run Sentiment_Analysis.py
enter your sentence I always feel a sense of joy and love when I spend time with my family, even if we're just doing somethin
g normal and average like watching TV. However, when someone in the family is angry or frustrated it can quickly turn a happy
 moment into a sad and tense one.
The percentage of positive words in the text is: 5.88235294117647 %
The percentage of negative words in the text is: 5.88235294117647 %
The percentage of neutral words in the text is: 3.9215686274509802 %

C:\Users\samah\OneDrive\Desktop\SE463_Project>coverage report
Name                     Stmts   Miss  Cover
--------------------------------------------
Sentiment_Analysis.py       37      8    78%
--------------------------------------------
TOTAL                       37      8    78%
```

Figure 1 – Tool Output for Statement Test for Test Case 04

b. Branch Coverage Report for Test Case 04:

```
C:\Users\samah\OneDrive\Desktop\SE463_Project>py -m coverage run --branch Sentiment_Analysis.py
enter your sentence I always feel a sense of joy and love when I spend time with my family, even if we're just doing somethin
g normal and average like watching TV. However, when someone in the family is angry or frustrated it can quickly turn a happy
 moment into a sad and tense one.
The percentage of positive words in the text is: 5.88235294117647 %
The percentage of negative words in the text is: 5.88235294117647 %
The percentage of neutral words in the text is: 3.9215686274509802 %

C:\Users\samah\OneDrive\Desktop\SE463_Project>coverage report
Name                     Stmts   Miss Branch BrPart  Cover
------------------------------------------------------------
Sentiment_Analysis.py       37      8     14      3    78%
------------------------------------------------------------
TOTAL                       37      8     14      3    78%
```

Figure 2 – Tool Output for Branch Test for Test Case 04

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 05:

a. Statement Coverage Report for Test Case 05:

```
C:\Users\Jana\Downloads>python -m coverage run TestingProject-SentimentAnalysis.py
enter your sentence I am so excited we are going to the zoo with my friend. I love monkeys and I am happy to see them in
 reality. However, she thinks it is a normal thing.
Positive 😻
The percentage of positive words in the text is: 9.090909090909092 %
The percentage of negative words in the text is: 0.0 %
The percentage of neutral words in the text is: 3.0303030303030303 %

C:\Users\Jana\Downloads>python -m coverage report -m
Name                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
TestingProject-SentimentAnalysis.py      37      9    76%   20, 32-41
-----------------------------------------------------------------
TOTAL                                    37      9    76%
```

Figure 1 – Tool Output for Statement Test for Test Case 05

b. Branch Coverage Report for Test Case 05:

```
C:\Users\Jana\Downloads>python -m coverage run --branch TestingProject-SentimentAnalysis.py
enter your sentence I am so excited we are going to the zoo with my friend. I love monkeys and I am happy to see them in
 reality. However, she thinks it is a normal thing.
Positive 😻
The percentage of positive words in the text is: 9.090909090909092 %
The percentage of negative words in the text is: 0.0 %
The percentage of neutral words in the text is: 3.0303030303030303 %

C:\Users\Jana\Downloads>python -m coverage report -m
Name                                Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------------------------------
TestingProject-SentimentAnalysis.py      37      9     14      2    71%   20, 32-41
-------------------------------------------------------------------------------
TOTAL                                    37      9     14      2    71%
```

Figure 2 – Tool Output for Branch Test for Test Case 05

Test Case 06:

a. Statement Coverage Report for Test Case 06:

```
C:\Users\Jana\Downloads>python -m coverage run TestingProject-SentimentAnalysis.py
enter your sentence I am so sad as my brother has an accident. It is an average accident and he is fine now. However, I
hate seeing him in such a situation like this.
The percentage of positive words in the text is: 0.0 %
The percentage of negative words in the text is: 6.451612903225806 %
The percentage of neutral words in the text is: 6.451612903225806 %

C:\Users\Jana\Downloads>python -m coverage report -m
Name                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
TestingProject-SentimentAnalysis.py      37      9    76%   18, 30-31, 33-38, 40-41
-----------------------------------------------------------------
TOTAL                                    37      9    76%
```

Figure 1 – Tool Output for Statement Test for Test Case 06

b. Branch Coverage Report for Test Case 06:

```
C:\Users\Jana\Downloads>python -m coverage run --branch TestingProject-SentimentAnalysis.py
enter your sentence I am so sad as my brother has an accident. It is an average accident and he is fine now. However, I
hate seeing him in such a situation like this.
The percentage of positive words in the text is: 0.0 %
The percentage of negative words in the text is: 6.451612903225806 %
The percentage of neutral words in the text is: 6.451612903225806 %

C:\Users\Jana\Downloads>python -m coverage report -m
Name                                Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------------------------------
TestingProject-SentimentAnalysis.py      37      9     14      4    75%   18, 30-31, 33-38, 40-41
-------------------------------------------------------------------------------
TOTAL                                    37      9     14      4    75%
```

Figure 2 – Tool Output for Branch Test for Test Case 06

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

4.4 Discussion of Manual Testing, and Automatic Testing Results

As we mention in the report, we apply the statement coverage tests for the same test cases manually and by opensource tool, we get these results.

Test Cases	Manual Testing	Opensource Testing Tool
Test Case 01	72.9%	73%
Test Case 02	81.08%	81%
Test Case 03	78.37%	78%
Test Case 04	78.37%	78%
Test Case 05	75.76%	76%
Test Case 06	75.67%	76%

Table 1- Result of Manual Testing & Opensource Testing Tool

We can notice that we get almost the same result; this means that our technique that we are following in conducting the test statement is correct.

According to brunch coverage we did the something, we test the same test case manually and by opensource tool, we get these results:

Test Cases	Manual Testing	Opensource Testing Tool
Test Case 01	50%	67%
Test Case 02	57.14%	75%
Test Case 03	64.28%	75%
Test Case 04	78.57%	78%
Test Case 05	57.14%	76%
Test Case 06	71.42%	71%

Table 2- Result of Manual Testing & Opensource Testing Tool

We found that there are differences in the results between the manual test and the test using an open-source tool. The reason for this difference is that the open-source tool calculated brunch coverage considering the statement coverage as well. The steps that followed by the tool to calculate brunch coverage are:

1. **Step1**: We will find: no_statements, no_missing, no_branches and no_missing_branches
2. **Step2:** Calculate the executed number of lines by using this formula:

   no_executed = no_statements - no_missing

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

3. **Step3:** Calculate the number of executed brunch by using this formula:

   no_executed_branches = no_branches - no_missing_branches

4. **Step4:**

   numerator = no_executed + no_executed_branches

   denominator = no_statements + no_branches

5. **Step5:** Now we calculate the cover by using this formula:

$$\text{Coverage} = \frac{numerator}{\text{denominator}} * 100$$

In the manual testing, we were applying our logical understanding with the following of the rules and instructions of the coverages, then we discover that there is no difference between our results and the automatic test coverage tool results. The rules of calculating the decision coverage were followed precisely.

CHAPTER 5: CONCLUSION

5.1 Conclusion

In conclusion, we applied our knowledge of Software Testing and Quality Assurance to our project. We began by writing a comprehensive test plan and designing test cases. To measure the code coverage, we utilized various techniques, including manual and automated testing and compare the results from manual testing to results from automated testing and they were identical. Throughout the project, we utilized the techniques and concepts we learned in our lectures and improved our understanding through hands-on application. Overall, this project allowed us to gain practical experience and reinforce our understanding of the course material.

5.2 References

[1] Hamilton, Thomas. “Code Coverage Tutorial: Branch, Statement, Decision, FSM.” Www.guru99.com, 20 Feb. 2020, www.guru99.com/code-coverage.html#:~:text=Branch%20Coverage%20is%20a%20white%20box%20testing%20method. Accessed 14 Apr. 2023.

[2] “Coverage.py,” Coverage.py - Coverage.py 5.5 documentation. [Online]. Available: https://coverage.readthedocs.io/en/coverage-5.5/#coverage-py. [Accessed: 27-Apr-2023].

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

[3] “What is software testing and how does it work?,” IBM. [Online]. Available: https://www.ibm.com/topics/software-testing. [Accessed: 01-May-2023].

[4] “The seven principles of testing,” www.boxuk.com, 02-Jul-2022. [Online]. Available: https://www.boxuk.com/insight/the-seven-principles-of-testing/. [Accessed: 01-May-2023].

[5] E. Kinsbruner, “The Complete Guide to different types of testing,” Perfecto . [Online]. Available: https://www.perfecto.io/resources/types-of-testing. [Accessed: 02-May-2023].

[6] D. Graham, Foundations of software testing ISTQB Certification, 1st ed. London: Thomson Learning, 2007.

[7] M. Simonova, “Council post: The distinction between testing and Quality Assurance in the software industry,” Forbes, 16-Aug-2022. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2022/08/12/the-distinction-between-testing-and-quality-assurance-in-the-software-industry/?sh=71465c50391d. [Accessed: 02-May-2023].

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

APPENDICES

1. Members Roles

To facilitate effective collaboration within our team, we have assigned specific roles to each team member, as outlined below:

Task	Jana	Salwa	Samah	Sana
Report				
1.0 Introduction		✓		
2.0 Software Testing	✓			
3.0 Testing Plan and Criteria				✓
3.1 Test Cases	✓	✓	✓	✓
4.0 Analyses and Discussion			✓	
5.0 Conclusion		✓		
Code and Tool				
Code	✓	✓	✓	✓
Tool Configuration	✓	✓	✓	✓

Table 1- Members Roles

2. Sentiment Analysis Code

Below is the source code for our sentiment analysis program, which is implemented in Python. The code has ٣٧ lines, excluding blank lines and comments.

```python
import random

text = input("enter your sentence ")

positive_words = ["happy", "joy", "love", "excited"]
negative_words = ["sad", "angry", "hate", "frustrated"]
neutral_words = ["okay", "fine", "normal", "average"]

positive_count = 0
negative_count = 0
neutral_count = 0

words = text.split(" ")

for word in words:
    if word.lower() in positive_words:
        positive_count += 1
    if word.lower() in negative_words:
        negative_count += 1
    if word.lower() in neutral_words:
        neutral_count += 1

# Generate a random emoji based on the sentiment
positive_emojis = ["😊", "😄", "😁", "😃"]
negative_emojis = ["😢", "😠", "😡", "😭"]
neutral_emojis = ["😐", "😶", "😑", "😒"]

```

Figure 1 - Sentiment Analysis Code Part1

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

```
if positive_count > negative_count and positive_count > neutral_count:
    emoji = random.choice(positive_emojis)
    print(f'Positive {emoji}')
elif negative_count > positive_count and negative_count > neutral_count:
    emoji = random.choice(negative_emojis)
    print(f"Negative {emoji}")
    jokes = ["Why was the math book sad? Because it had too many problems.",
             "Why don't scientists trust atoms? Because they make up everything.",
             "Why do programmers prefer dark mode? Because light attracts bugs."]
    print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
elif neutral_count > positive_count and neutral_count > negative_count:
    emoji = random.choice(neutral_emojis)
    print(f"Neutral {emoji}")

total_words = len(words)

percentage_positive = (positive_count / total_words) * 100
percentage_negative = (negative_count / total_words) * 100
percentage_neutral = (neutral_count / total_words) * 100

print("The percentage of positive words in the text is:", percentage_positive,"%")
print("The percentage of negative words in the text is:", percentage_negative,"%")
print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 2 - Sentiment Analysis Code Part2

3. Coverage Report

Test Case 01:

a. Statement Coverage Report for Test Case 01

Coverage for **Sentiment_Analysis.py**: 73%

37 statements | 27 run | 10 missing | 0 excluded

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-15 13:03 +0300

```
1  import random
2
3  text = input("enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😃", "🤩"]
25 negative_emojis = ["😢", "😠", "😞", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35              "Why don't scientists trust atoms? Because they make up everything.",
36              "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure ١ – Statement Report for Test Case 01

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 01:

Coverage for **Sentiment_Analysis.py**: 67%

37 statements | 27 run | 10 missing | 0 excluded | 3 partial

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-15 13:21 +0300

```
1  import random
2
3  text = input("enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😃", "🤩"]
25 negative_emojis = ["😢", "😠", "😡", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35                  "Why don't scientists trust atoms? Because they make up everything.",
36                  "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 2– Branch Report for Test Case 01

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 02:

a. Statement Coverage Report for Test Case 02:

Coverage for **Sentiment_Analysis.py**: 81%

37 statements | 30 run | 7 missing | 0 excluded

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-15 13:11 +0300

```
1  import random
2
3  text = input("enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😃", "🤩"]
25 negative_emojis = ["😢", "😠", "😡", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35              "Why don't scientists trust atoms? Because they make up everything.",
36              "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 1 – Statement Report for Test Case 02

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 02:

Coverage for **Sentiment_Analysis.py**: 75%

37 statements | 30 run | 7 missing | 0 excluded | 4 partial

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-15 13:15 +0300

```
1  import random
2
3  text = input("enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😻", "🤩"]
25 negative_emojis = ["😢", "😠", "😡", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35              "Why don't scientists trust atoms? Because they make up everything.",
36              "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 2 – Branch Report for Test Case 02

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 03:

a. Statement Coverage Report for Test Case 03:

SentimentAnalysis.py: 78% 29 8 0

```
1  import random
2
3  text = input("Enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😃", "🤩"]
25 negative_emojis = ["😢", "😠", "😡", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35              "Why don't scientists trust atoms? Because they make up everything.",
36              "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 1 – Statement Report for Test Case 03

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 03:

SentimentAnalysis.py: 75% 29 8 0 5

```
1  import random
2
3  text = input("Enter your sentence ")
4
5  positive_words = ["happy", "joy", "love", "excited"]
6  negative_words = ["sad", "angry", "hate", "frustrated"]
7  neutral_words = ["okay", "fine", "normal", "average"]
8
9  positive_count = 0
10 negative_count = 0
11 neutral_count = 0
12
13 words = text.split(" ")
14
15 for word in words:
16     if word.lower() in positive_words:
17         positive_count += 1
18     if word.lower() in negative_words:
19         negative_count += 1
20     if word.lower() in neutral_words:
21         neutral_count += 1
22
23 # Generate a random emoji based on the sentiment
24 positive_emojis = ["😊", "😍", "😘", "🤩"]
25 negative_emojis = ["😢", "😠", "😡", "😤"]
26 neutral_emojis = ["😐", "😑", "😶", "🙄"]
27
28 if positive_count > negative_count and positive_count > neutral_count:
29     emoji = random.choice(positive_emojis)
30     print(f'Positive {emoji}')
31 elif negative_count > positive_count and negative_count > neutral_count:
32     emoji = random.choice(negative_emojis)
33     print(f"Negative {emoji}")
34     jokes = ["Why was the math book sad? Because it had too many problems.",
35                 "Why don't scientists trust atoms? Because they make up everything.",
36                 "Why do programmers prefer dark mode? Because light attracts bugs."]
37     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
38 elif neutral_count > positive_count and neutral_count > negative_count:
39     emoji = random.choice(neutral_emojis)
40     print(f"Neutral {emoji}")
41
42 total_words = len(words)
43
44 percentage_positive = (positive_count / total_words) * 100
45 percentage_negative = (negative_count / total_words) * 100
46 percentage_neutral = (neutral_count / total_words) * 100
47
48 print("The percentage of positive words in the text is:", percentage_positive,"%")
49 print("The percentage of negative words in the text is:", percentage_negative,"%")
50 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
```

Figure 2 – Branch Report for Test Case 03

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 04:

a. Statement Coverage Report for Test Case 04:

Coverage for **Sentiment_Analysis.py**: 78%

37 statements | 29 run | 8 missing | 0 excluded

« prev ^ index » next coverage.py v7.2.3, created at 2023-05-06 23:20 +0300

```
1  import random
2
3
4  text = input("enter your sentence ")
5
6  positive_words = ["happy", "joy", "love", "excited"]
7  negative_words = ["sad", "angry", "hate", "frustrated"]
8  neutral_words = ["okay", "fine", "normal", "average"]
9
10 positive_count = 0
11 negative_count = 0
12 neutral_count = 0
13
14 words = text.split(" ")
15
16 for word in words:
17     if word.lower() in positive_words:
18         positive_count += 1
19     if word.lower() in negative_words:
20         negative_count += 1
21     if word.lower() in neutral_words:
22         neutral_count += 1
23
24 # Generate a random emoji based on the sentiment
25 positive_emojis = ["😊", "😍", "😘", "🤩"]
26 negative_emojis = ["😞", "😢", "😡", "😤"]
27 neutral_emojis = ["😐", "😑", "😶", "🙄"]
28
29 if positive_count > negative_count and positive_count > neutral_count:
30     emoji = random.choice(positive_emojis)
31     print(f'Positive {emoji}')
32 elif negative_count > positive_count and negative_count > neutral_count:
33     emoji = random.choice(negative_emojis)
34     print(f"Negative {emoji}")
35     jokes = ["Why was the math book sad? Because it had too many problems.",
36              "Why don't scientists trust atoms? Because they make up everything.",
37              "Why do programmers prefer dark mode? Because light attracts bugs."]
38     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
39 elif neutral_count > positive_count and neutral_count > negative_count:
40     emoji = random.choice(neutral_emojis)
41     print(f"Neutral {emoji}")
42
43 total_words = len(words)
44
45 percentage_positive = (positive_count / total_words) * 100
46 percentage_negative = (negative_count / total_words) * 100
47 percentage_neutral = (neutral_count / total_words) * 100
48
49 print("The percentage of positive words in the text is:", percentage_positive,"%")
50 print("The percentage of negative words in the text is:", percentage_negative,"%")
51 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
52
53
```

Figure 1 – Statement Report for Test Case 04

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 04:

Coverage for **Sentiment_Analysis.py**: 78%

37 statements | 29 run | 8 missing | 0 excluded | 3 partial

« prev ^ index » next coverage.py v7.2.3, created at 2023-05-06 23:31 +0300

```
1  import random
2
3
4  text = input("enter your sentence ")
5
6  positive_words = ["happy", "joy", "love", "excited"]
7  negative_words = ["sad", "angry", "hate", "frustrated"]
8  neutral_words = ["okay", "fine", "normal", "average"]
9
10 positive_count = 0
11 negative_count = 0
12 neutral_count = 0
13
14 words = text.split(" ")
15
16 for word in words:
17     if word.lower() in positive_words:
18         positive_count += 1
19     if word.lower() in negative_words:
20         negative_count += 1
21     if word.lower() in neutral_words:
22         neutral_count += 1
23
24 # Generate a random emoji based on the sentiment
25 positive_emojis = ["😊", "😍", "😘", "🥰"]
26 negative_emojis = ["😢", "😠", "😡", "😤"]
27 neutral_emojis = ["😐", "😑", "😶", "🙄"]
28
29 if positive_count > negative_count and positive_count > neutral_count:
30     emoji = random.choice(positive_emojis)
31     print(f'Positive {emoji}')
32 elif negative_count > positive_count and negative_count > neutral_count:
33     emoji = random.choice(negative_emojis)
34     print(f"Negative {emoji}")
35     jokes = ["Why was the math book sad? Because it had too many problems.",
36              "Why don't scientists trust atoms? Because they make up everything.",
37              "Why do programmers prefer dark mode? Because light attracts bugs."]
38     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
39 elif neutral_count > positive_count and neutral_count > negative_count:
40     emoji = random.choice(neutral_emojis)
41     print(f"Neutral {emoji}")
42
43 total_words = len(words)
44
45 percentage_positive = (positive_count / total_words) * 100
46 percentage_negative = (negative_count / total_words) * 100
47 percentage_neutral = (neutral_count / total_words) * 100
48
49 print("The percentage of positive words in the text is:", percentage_positive,"%")
50 print("The percentage of negative words in the text is:", percentage_negative,"%")
51 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
52
53
```

Figure 2 – Branch Report for Test Case 04

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 05:

a. Statement Coverage Report for Test Case 05:

TestingProject-SentimentAnalysis.py: 76% 28 9 0

```
1  import random
2
3
4  text = input("enter your sentence ")
5
6  positive_words = ["happy", "joy", "love", "excited"]
7  negative_words = ["sad", "angry", "hate", "frustrated"]
8  neutral_words = ["okay", "fine", "normal", "average"]
9
10 positive_count = 0
11 negative_count = 0
12 neutral_count = 0
13
14 words = text.split(" ")
15
16 for word in words:
17     if word.lower() in positive_words:
18         positive_count += 1
19     if word.lower() in negative_words:
20         negative_count += 1
21     if word.lower() in neutral_words:
22         neutral_count += 1
23
24 # Generate a random emoji based on the sentiment
25 positive_emojis = ["😊", "😍", "😻", "🤩"]
26 negative_emojis = ["😞", "😠", "🤬", "😤"]
27 neutral_emojis = ["😐", "😑", "😶", "🙄"]
28
29 if positive_count > negative_count and positive_count > neutral_count:
30     emoji = random.choice(positive_emojis)
31     print(f'Positive {emoji}')
32 elif negative_count > positive_count and negative_count > neutral_count:
33     emoji = random.choice(negative_emojis)
34     print(f"Negative {emoji}")
35     jokes = ["Why was the math book sad? Because it had too many problems.",
36              "Why don't scientists trust atoms? Because they make up everything.",
37              "Why do programmers prefer dark mode? Because light attracts bugs."]
38     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
39 elif neutral_count > positive_count and neutral_count > negative_count:
40     emoji = random.choice(neutral_emojis)
41     print(f"Neutral {emoji}")
42
43 total_words = len(words)
44
45 percentage_positive = (positive_count / total_words) * 100
46 percentage_negative = (negative_count / total_words) * 100
47 percentage_neutral = (neutral_count / total_words) * 100
48
49 print("The percentage of positive words in the text is:", percentage_positive,"%")
50 print("The percentage of negative words in the text is:", percentage_negative,"%")
51 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
52
```

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-26 17:44 +0300

Figure 1 – Statement Report for Test Case 05

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 05:

TestingProject-SentimentAnalysis.py: 71% | 28 | 9 | 0 | 2

```
1  import random
2
3
4  text = input("enter your sentence ")
5
6  positive_words = ["happy", "joy", "love", "excited"]
7  negative_words = ["sad", "angry", "hate", "frustrated"]
8  neutral_words = ["okay", "fine", "normal", "average"]
9
10 positive_count = 0
11 negative_count = 0
12 neutral_count = 0
13
14 words = text.split(" ")
15
16 for word in words:
17     if word.lower() in positive_words:
18         positive_count += 1
19     if word.lower() in negative_words:
20         negative_count += 1
21     if word.lower() in neutral_words:
22         neutral_count += 1
23
24 # Generate a random emoji based on the sentiment
25 positive_emojis = ["😊", "😍", "😘", "🤩"]
26 negative_emojis = ["😢", "😠", "🤬", "😤"]
27 neutral_emojis = ["😐", "😑", "😶", "🙄"]
28
29 if positive_count > negative_count and positive_count > neutral_count:
30     emoji = random.choice(positive_emojis)
31     print(f'Positive {emoji}')
32 elif negative_count > positive_count and negative_count > neutral_count:
33     emoji = random.choice(negative_emojis)
34     print(f"Negative {emoji}")
35     jokes = ["Why was the math book sad? Because it had too many problems.",
36              "Why don't scientists trust atoms? Because they make up everything.",
37              "Why do programmers prefer dark mode? Because light attracts bugs."]
38     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
39 elif neutral_count > positive_count and neutral_count > negative_count:
40     emoji = random.choice(neutral_emojis)
41     print(f"Neutral {emoji}")
42
43 total_words = len(words)
44
45 percentage_positive = (positive_count / total_words) * 100
46 percentage_negative = (negative_count / total_words) * 100
47 percentage_neutral = (neutral_count / total_words) * 100
48
49 print("The percentage of positive words in the text is:", percentage_positive,"%")
50 print("The percentage of negative words in the text is:", percentage_negative,"%")
51 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
52
```

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-26 17:46 +0300

Figure 2 – Branch Report for Test Case 05

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

Test Case 06:

a. Statement Coverage Report for Test Case 06:

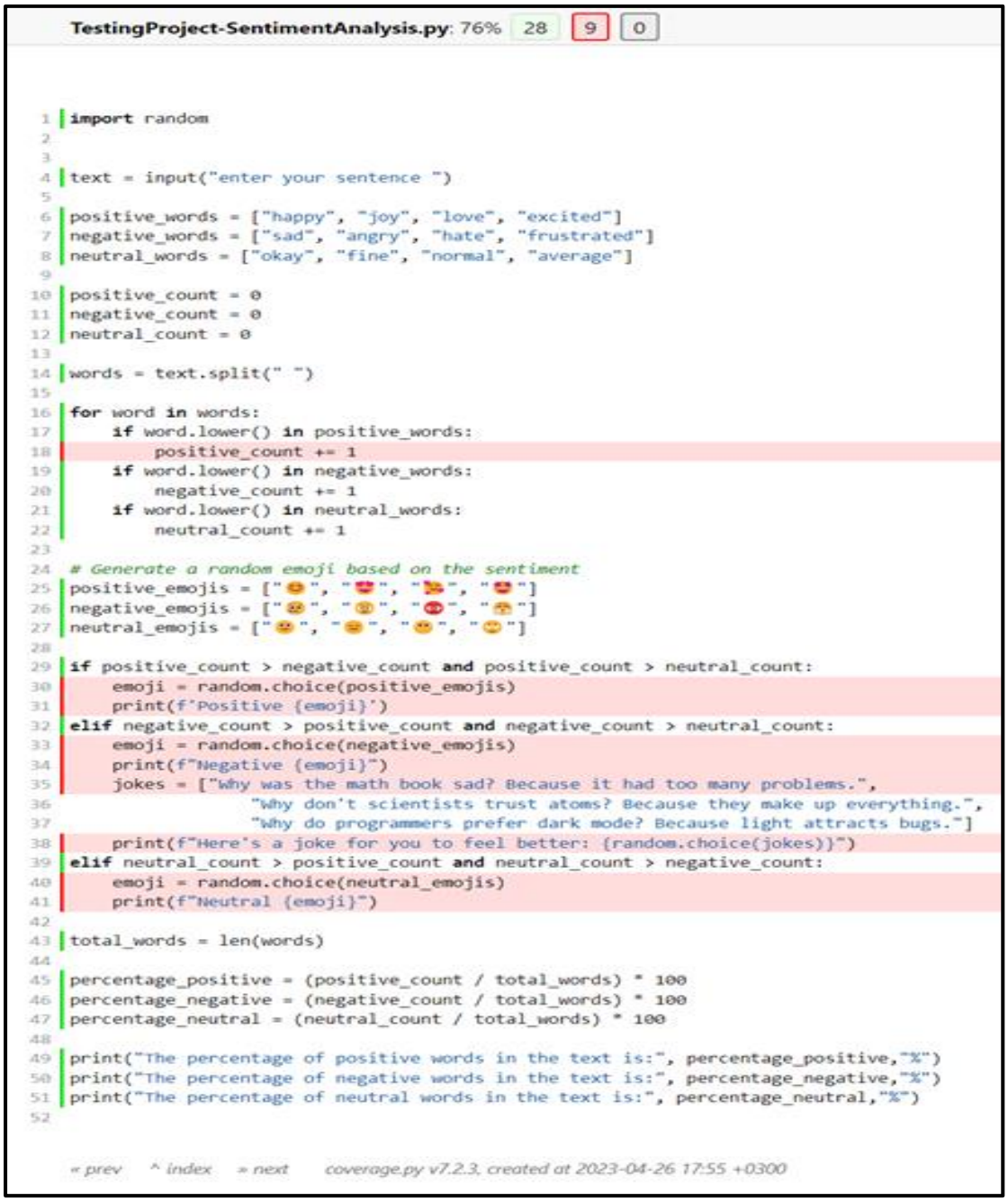

Figure 1 – Statement Report for Test Case 06

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*

b. Branch Coverage Report for Test Case 06:

TestingProject-SentimentAnalysis.py: 75% | 28 | 9 | 0 | 4

```
1  import random
2
3
4  text = input("enter your sentence ")
5
6  positive_words = ["happy", "joy", "love", "excited"]
7  negative_words = ["sad", "angry", "hate", "frustrated"]
8  neutral_words = ["okay", "fine", "normal", "average"]
9
10 positive_count = 0
11 negative_count = 0
12 neutral_count = 0
13
14 words = text.split(" ")
15
16 for word in words:
17     if word.lower() in positive_words:
18         positive_count += 1
19     if word.lower() in negative_words:
20         negative_count += 1
21     if word.lower() in neutral_words:
22         neutral_count += 1
23
24 # Generate a random emoji based on the sentiment
25 positive_emojis = ["😊", "😍", "😻", "😽"]
26 negative_emojis = ["😢", "😠", "🤡", "😤"]
27 neutral_emojis = ["😐", "😑", "😶", "🙄"]
28
29 if positive_count > negative_count and positive_count > neutral_count:
30     emoji = random.choice(positive_emojis)
31     print(f'Positive {emoji}')
32 elif negative_count > positive_count and negative_count > neutral_count:
33     emoji = random.choice(negative_emojis)
34     print(f"Negative {emoji}")
35     jokes = ["Why was the math book sad? Because it had too many problems.",
36              "Why don't scientists trust atoms? Because they make up everything.",
37              "Why do programmers prefer dark mode? Because light attracts bugs."]
38     print(f"Here's a joke for you to feel better: {random.choice(jokes)}")
39 elif neutral_count > positive_count and neutral_count > negative_count:
40     emoji = random.choice(neutral_emojis)
41     print(f"Neutral {emoji}")
42
43 total_words = len(words)
44
45 percentage_positive = (positive_count / total_words) * 100
46 percentage_negative = (negative_count / total_words) * 100
47 percentage_neutral = (neutral_count / total_words) * 100
48
49 print("The percentage of positive words in the text is:", percentage_positive,"%")
50 print("The percentage of negative words in the text is:", percentage_negative,"%")
51 print("The percentage of neutral words in the text is:", percentage_neutral,"%")
52
```

« prev ^ index » next coverage.py v7.2.3, created at 2023-04-26 17:57 +0300

Figure 2 – Branch Report for Test Case 06

** we refer to the lines number in figure 1 & 2 in the appendices in all test cases*