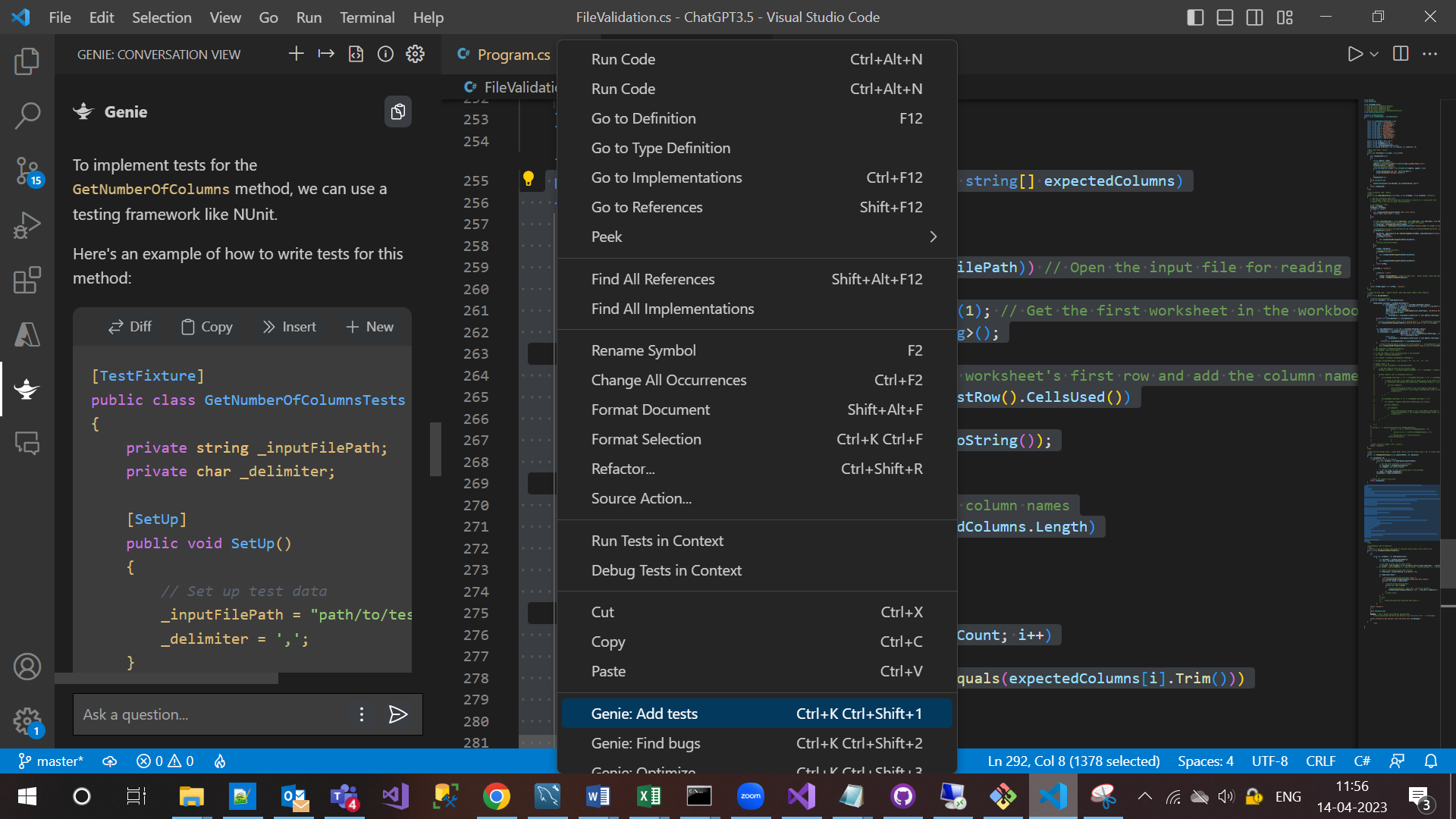
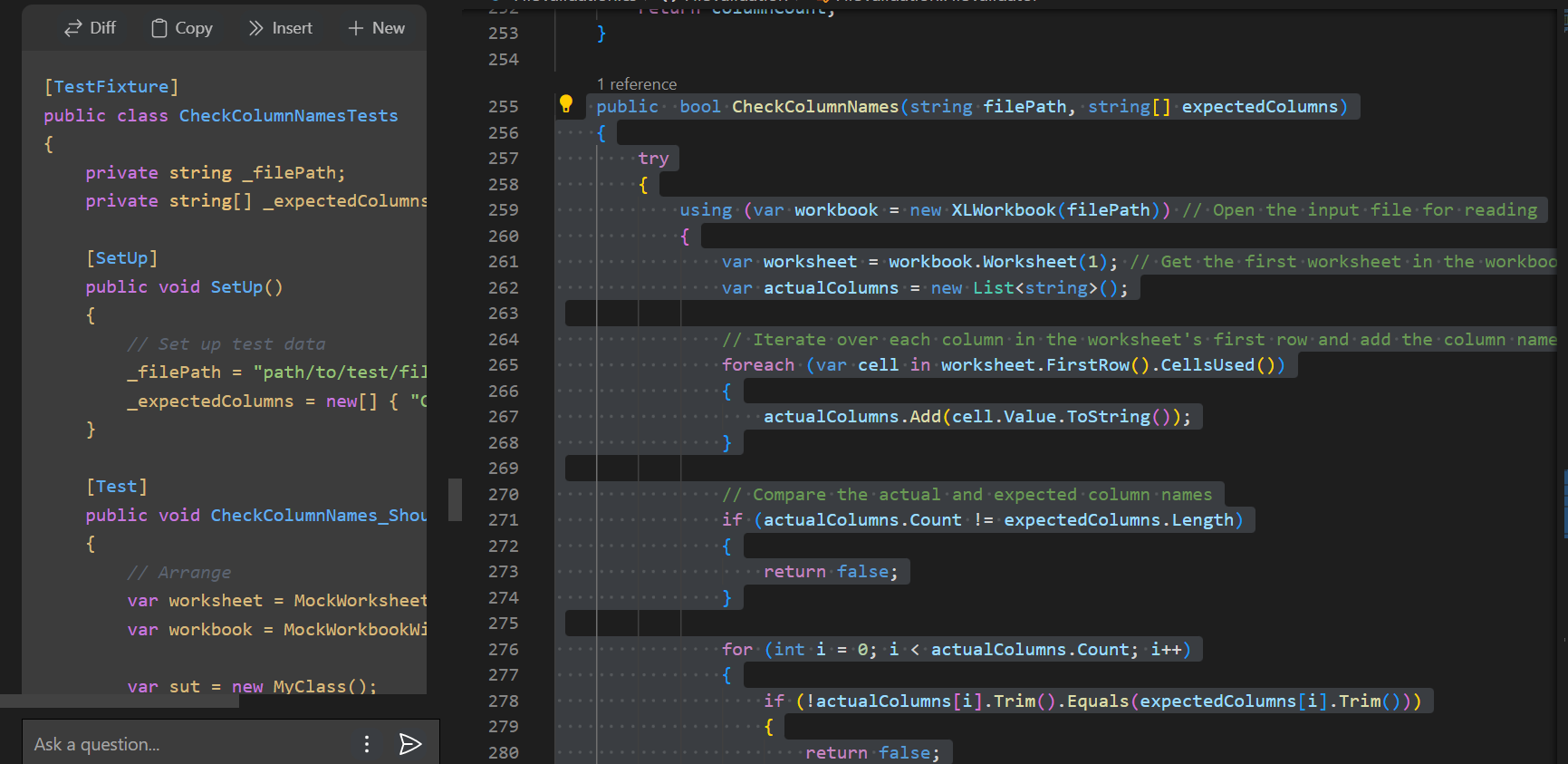
To implement tests for the CheckColumnNames method, we can use a testing framework like NUnit.

Here's an example of how to write tests for this method:





Set up the testing environment:

Declare and initialize variables for the filePath, expectedColumns.

Create an instance of the class that contains the method to be tested.

Test the CheckColumnNames() method:

Provide an input Excel file containing columns matching the expected column names.

Invoke the CheckColumnNames() method with the input file path and a string array of the expected column names.

Assert that the returned value is true.

Test the CheckColumnNames() method with an input file that has a different number of columns than expected:

Provide an input Excel file containing a different number of columns than expected.

Invoke the CheckColumnNames() method with the input file path and a string array of the expected column names.

Assert that the returned value is false.

Test the CheckColumnNames() method with an input file that has different column names than expected:

Provide an input Excel file containing column names that do not match the expected column names.

Invoke the CheckColumnNames() method with the input file path and a string array of the expected column names.

Assert that the returned value is false.

Test the exception handling of the CheckColumnNames() method:

Provide an invalid input file path.

Invoke the CheckColumnNames() method with the input file path and a string array of the expected column names.

Assert that the returned value is false.

Assert that an error message was written to the console.

[TestClass]

public class ExcelProcessorTests

{

// Step 1: Set up the testing environment

private string filePath = "input.xlsx";

private string[] expectedColumns = new string[] { "Name", "Email", "Phone" };

private ExcelProcessor excelProcessor;

[TestInitialize]

public void Setup()

{

excelProcessor = new ExcelProcessor();

}

[TestMethod]

public void CheckColumnNames\_ReturnsTrueForMatchingFile()

{

// Step 2: Provide an input file with matching columns

// and invoke the method

var result = excelProcessor.CheckColumnNames(filePath, expectedColumns);

// Assert that the method returned true

Assert.IsTrue(result);

}

[TestMethod]

public void CheckColumnNames\_ReturnsFalseForWrongNumberOfColumns()

{

// Step 3: Provide an input file with a different number of columns

// than expected and invoke the method

var result = excelProcessor.CheckColumnNames("input\_wrongcolumns.xlsx", expectedColumns);

// Assert that the method returned false

Assert.IsFalse(result);

}

[TestMethod]

public void CheckColumnNames\_ReturnsFalseForWrongColumnNames()

{

// Step 4: Provide an input file with different column names

// than expected and invoke the method

var result = excelProcessor.CheckColumnNames("input\_wrongnames.xlsx", expectedColumns);

// Assert that the method returned false

Assert.IsFalse(result);

}

[TestMethod]

public void CheckColumnNames\_ReturnsFalseForInvalidFile()

{

// Step 5: Provide an invalid input file path and invoke the method

var result = excelProcessor.CheckColumnNames("nonexistent.xlsx", expectedColumns);

// Assert that the method returned false and an error message was written to the console

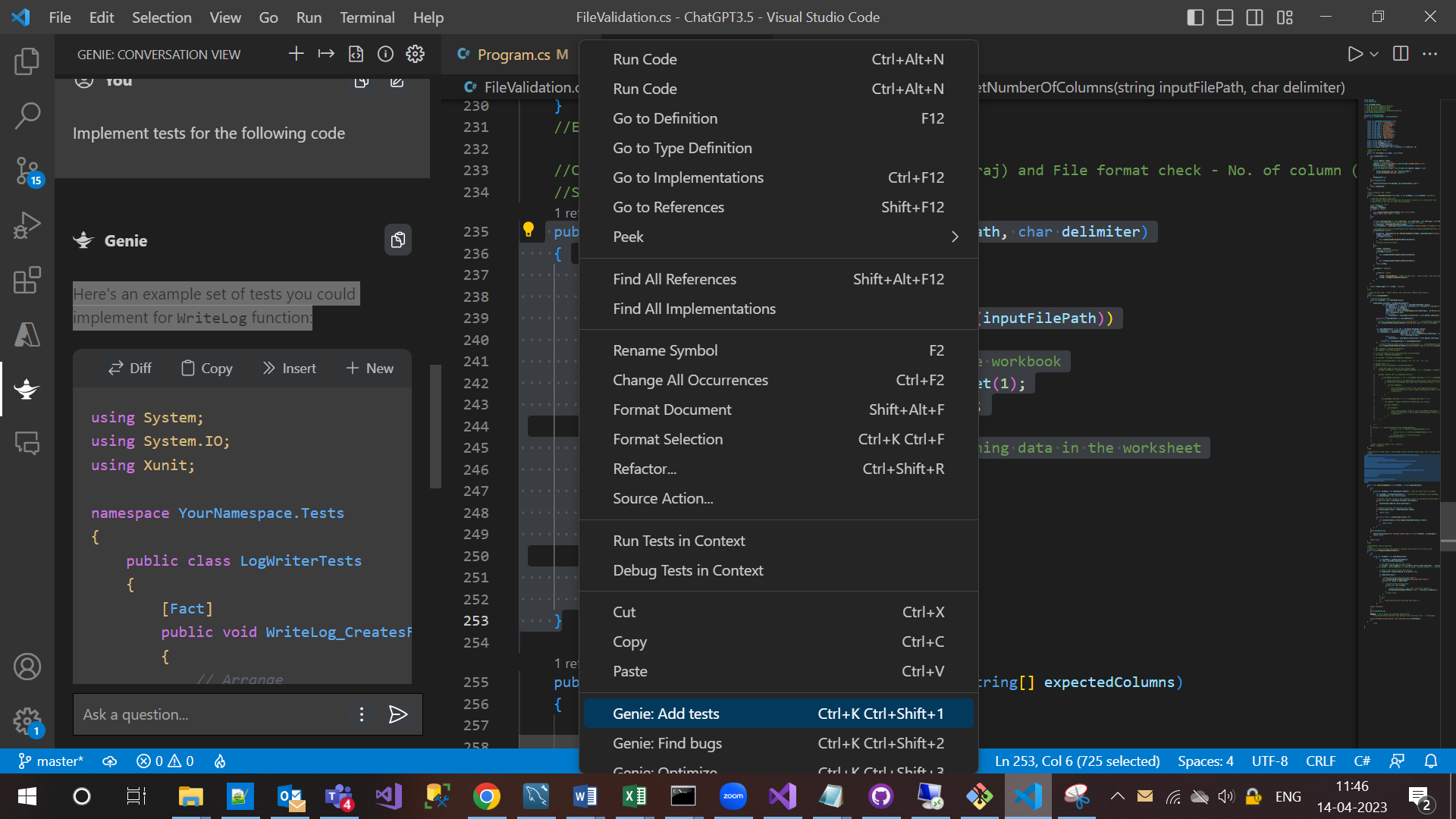
Assert.IsFalse(result);

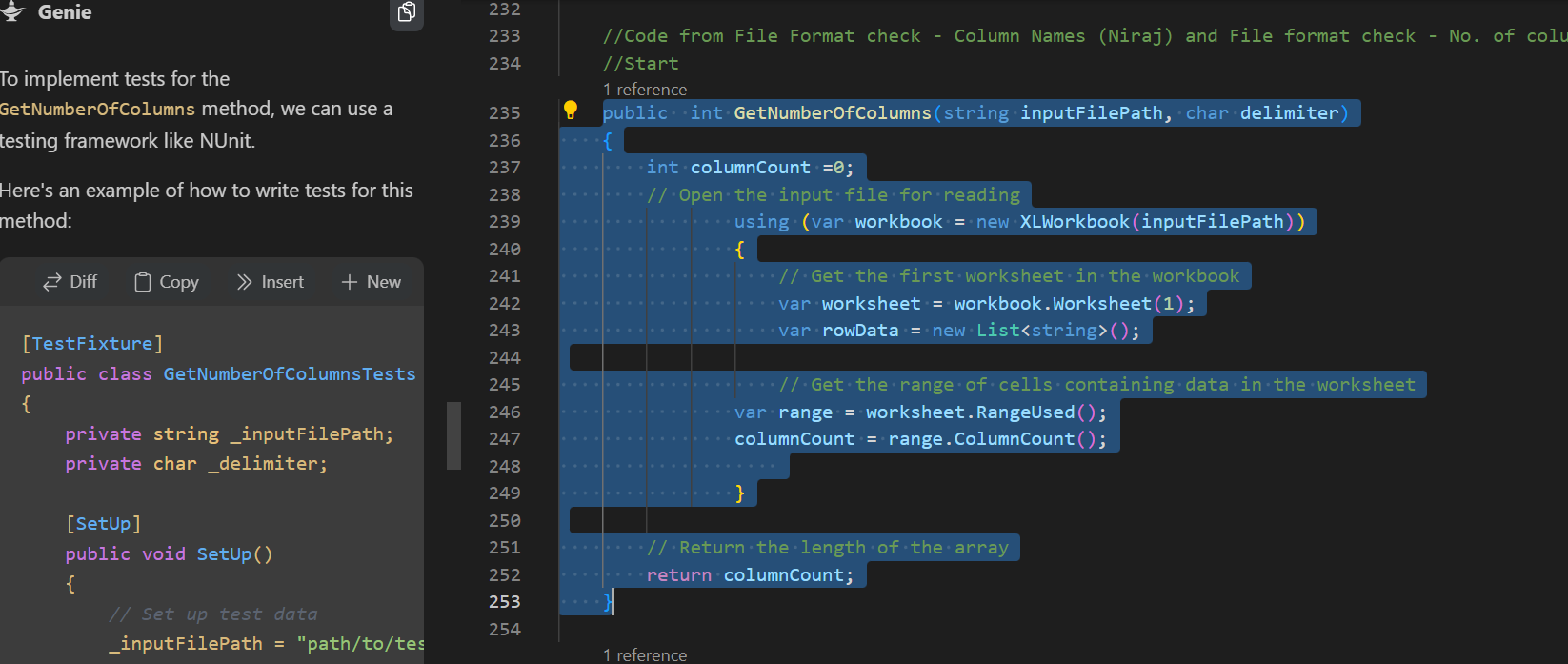
StringAssert.Contains(Console.Out.ToString(), "Error checking column names for file nonexistent.xlsx:");

}

}

--------------------------------------------- GetNumberOfColumns---------------------------------------------------------





Set up the testing environment:

Declare and initialize variables for the inputFilePath and delimiter.

Create an instance of the class that contains the method to be tested.

Test the GetNumberOfColumns() method:

Provide an input file containing columns separated by the specified delimiter.

Invoke the GetNumberOfColumns() method with the input file path and delimiter character as arguments.

Assert that the returned value is equal to the expected number of columns in the file.

Test the GetNumberOfColumns() method with an input file that does not contain any data:

Provide an input file that does not contain any data.

Invoke the GetNumberOfColumns() method with the input file path and delimiter character as arguments.

Assert that the returned value is 0.

Test the exception handling of the GetNumberOfColumns() method:

Provide an invalid input file path.

Invoke the GetNumberOfColumns() method with the input file path and delimiter character as arguments.

Assert that an error message was written to the console.

[TestClass]

public class ExcelProcessorTests

{

// Step 1: Set up the testing environment

private string inputFilePath = "input.csv";

private char delimiter = ',';

private ExcelProcessor excelProcessor;

[TestInitialize]

public void Setup()

{

excelProcessor = new ExcelProcessor();

}

[TestMethod]

public void GetNumberOfColumns\_ReturnsCorrectCountForFile()

{

// Step 2: Provide an input file and invoke the method

var result = excelProcessor.GetNumberOfColumns(inputFilePath, delimiter);

// Assert that the method returned the correct number of columns

Assert.AreEqual(3, result);

}

[TestMethod]

public void GetNumberOfColumns\_ReturnsZeroForEmptyFile()

{

// Step 3: Provide an empty input file and invoke the method

var result = excelProcessor.GetNumberOfColumns("input\_empty.csv", delimiter);

// Assert that the method returned 0

Assert.AreEqual(0, result);

}

[TestMethod]

public void GetNumberOfColumns\_ThrowsExceptionForInvalidFile()

{

// Step 4: Provide an invalid input file path and invoke the method

try

{

var result = excelProcessor.GetNumberOfColumns("nonexistent.csv", delimiter);

}

catch (Exception ex)

{

// Assert that an error message was thrown

StringAssert.Contains(ex.Message, "Error opening file nonexistent.csv:");

return;

}

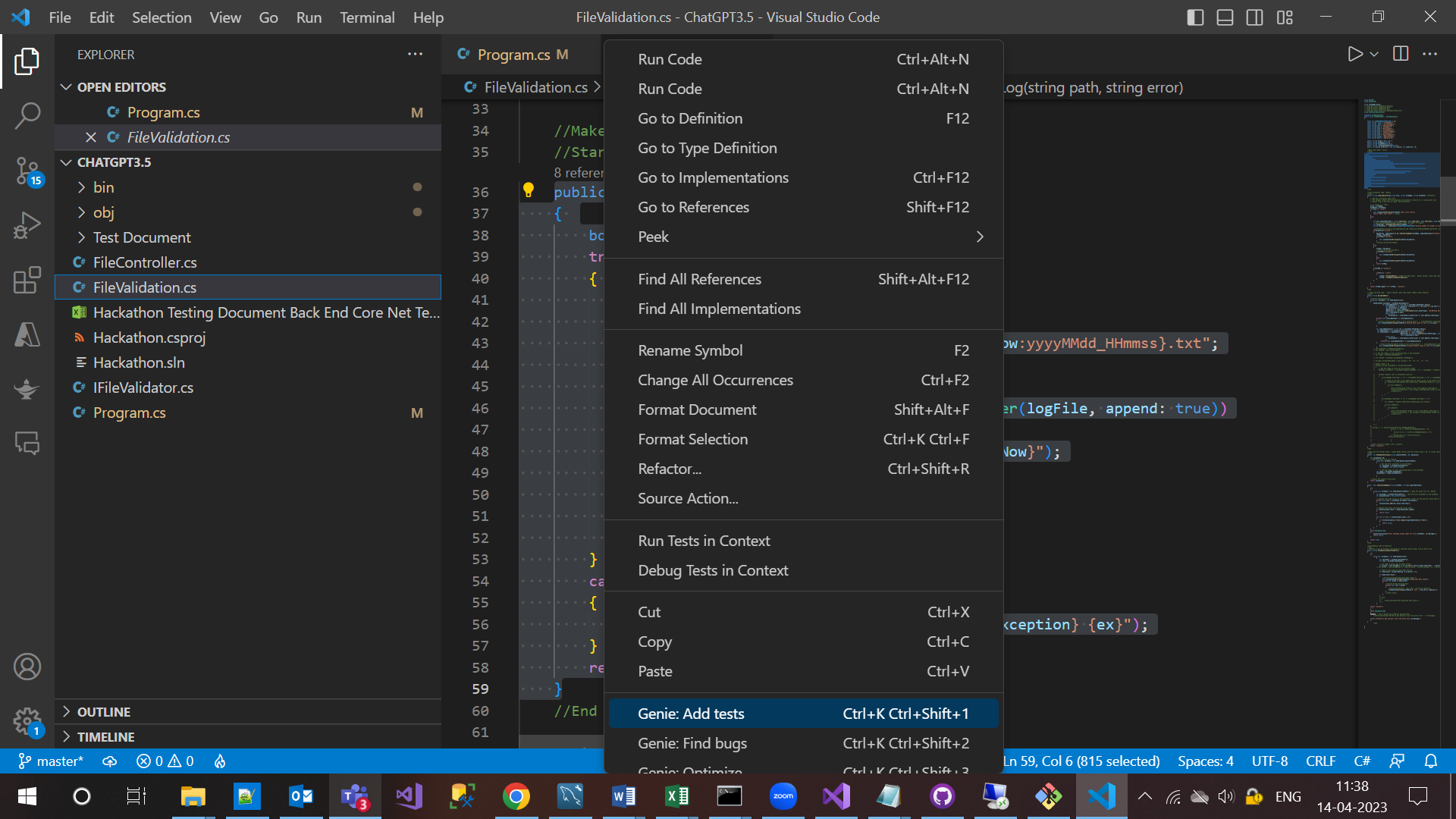
Assert.Fail("Expected exception was not thrown.");

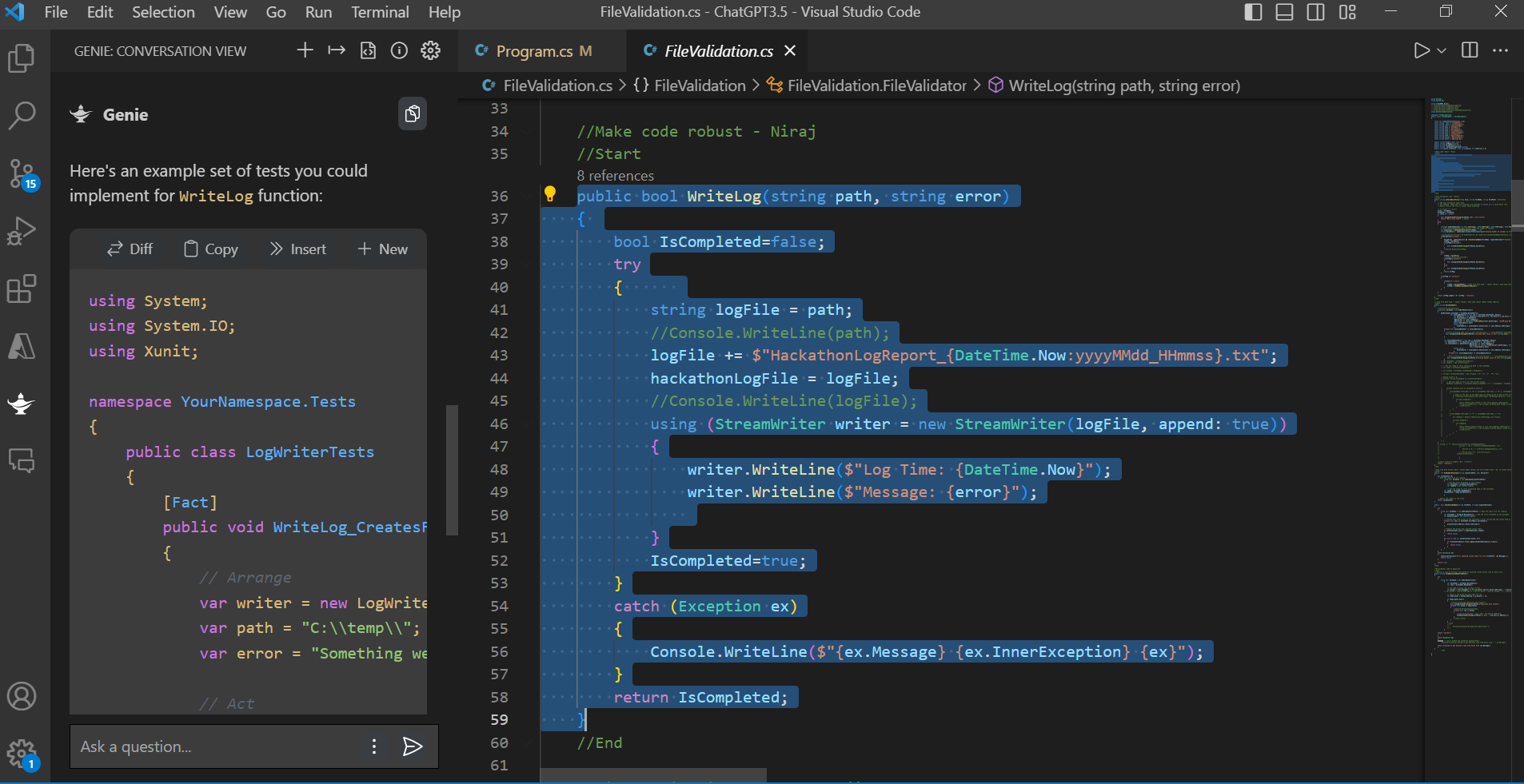
}

}

-----WriteLog------------------------------------------------------------------

Here's an example set of tests you could implement for WriteLog function:





Set up the testing environment:

Declare and initialize a variable for the log file path.

Create an instance of the class that contains the method to be tested.

Create or provide sample errors.

Test the WriteLog() method:

Invoke the WriteLog() method with a valid log file path and an error message as arguments.

Assert that the method returns true.

Assert that the log file was created and contains the correct error message.

Test the exception handling of the WriteLog() method:

Provide an invalid file path.

Invoke the WriteLog() method with the invalid file path and an error message as arguments.

Assert that the method returns false.

[TestClass]

public class LoggerTests

{

// Step 1: Set up the testing environment

private string logsPath = "logs/";

private Logger logger;

[TestInitialize]

public void Setup()

{

logger = new Logger();

}

[TestMethod]

public void WriteLog\_CreatesLogFileAndReturnsTrue()

{

// Step 2: Provide a valid log file path and invoke the method

var result = logger.WriteLog(logsPath + "test\_log.txt", "Test error");

// Assert that the method returned true

Assert.IsTrue(result);

// Assert that the log file was created and contains the correct message

string[] lines = File.ReadAllLines(logsPath + "test\_log.txt");

StringAssert.Contains(lines[0], "Log Time:");

StringAssert.Contains(lines[1], "Message:");

StringAssert.Contains(lines[1], "Test error");

}

[TestMethod]

public void WriteLog\_ReturnsFalseForInvalidPath()

{

// Step 3: Provide an invalid log file path and invoke the method

var result = logger.WriteLog("", "Test error");

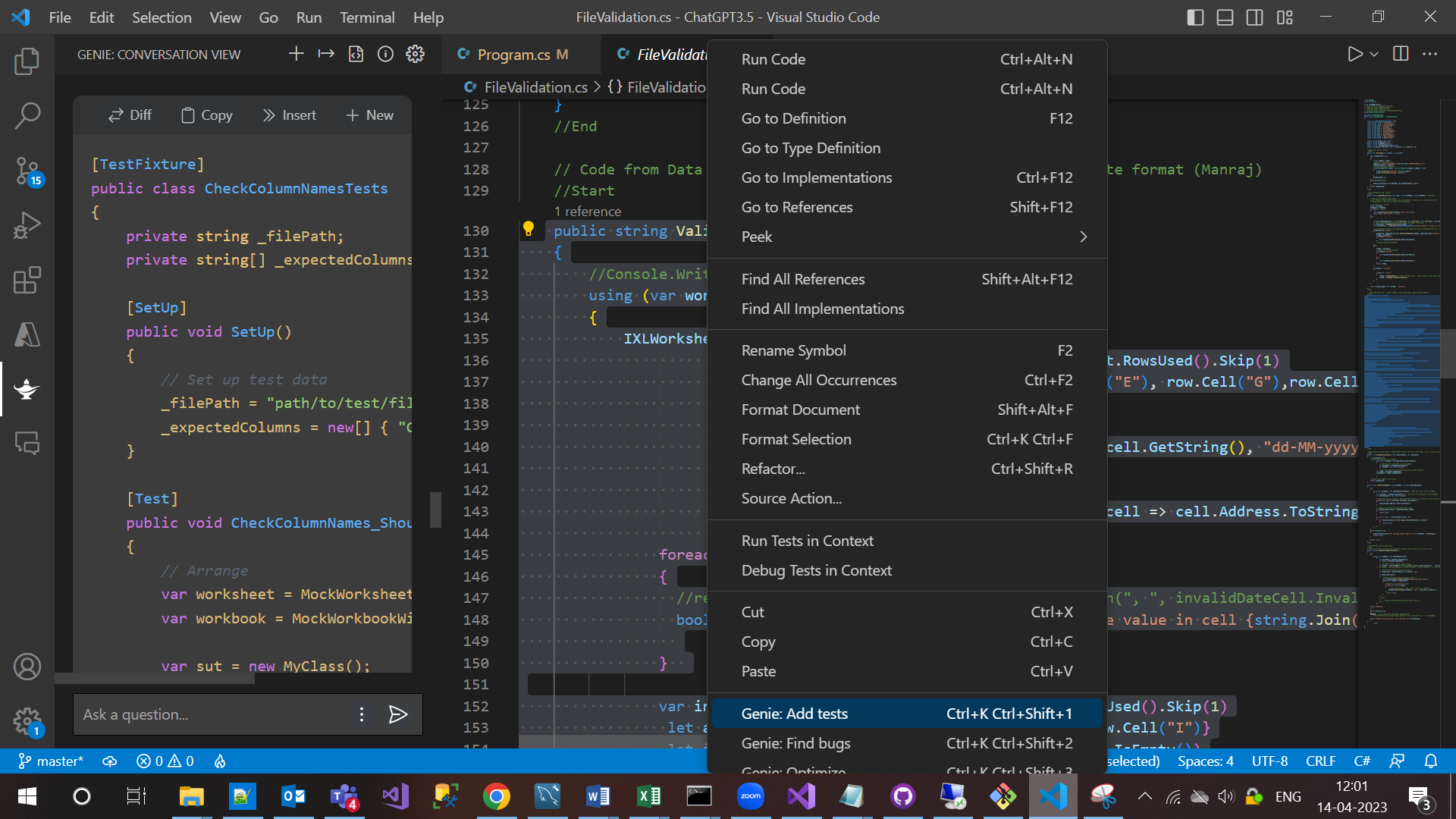
// Assert that the method returned false

Assert.IsFalse(result);

}

}

------------------------ ValidateDate------------------------



Set up the testing environment:

Declare and initialize variables for the file path and file name.

Create an instance of the class that contains the method to be tested.

Create or get sample CSV files containing valid and invalid date and amount values.

Test the ValidateDate() method:

Invoke the ValidateDate() method with the input file path as argument.

Assert that the method returns "success" if all dates and amounts are valid.

Assert that the method writes log messages to file if there are any invalid dates or amounts.

Test the exception handling of the ValidateDate() method:

Provide an invalid file path.

Invoke the ValidateDate() method with the input file path as argument.

Assert that an error message was written to the console.

[TestClass]

public class ExcelProcessorTests

{

// Step 1: Set up the testing environment

private string validFilePath = "valid.csv";

private string invalidDatesFilePath = "invalid\_dates.csv";

private string invalidAmountsFilePath = "invalid\_amounts.csv";

private string invalidFilePath = "nonexistent.csv";

private ExcelProcessor excelProcessor;

private string logsPath = "logs.txt";

[TestInitialize]

public void Setup()

{

excelProcessor = new ExcelProcessor();

}

[TestMethod]

public void ValidateDate\_ReturnsSuccessForValidFile()

{

// Step 2: Provide a valid input file and invoke the method

var result = excelProcessor.ValidateDate(validFilePath);

// Assert that the method returned "success"

Assert.AreEqual("success", result);

}

[TestMethod]

public void ValidateDate\_WritesLogForInvalidDates()

{

// Step 2: Provide a file with invalid dates and invoke the method

var result = excelProcessor.ValidateDate(invalidDatesFilePath);

// Assert that the method wrote log messages to file

string[] logs = File.ReadAllLines(logsPath);

Assert.IsTrue(logs.Length > 0);

Assert.IsTrue(logs[logs.Length - 1].Contains("Invalid date value"));

}

[TestMethod]

public void ValidateDate\_WritesLogForInvalidAmounts()

{

// Step 2: Provide a file with invalid amounts and invoke the method

var result = excelProcessor.ValidateDate(invalidAmountsFilePath);

// Assert that the method wrote log messages to file

string[] logs = File.ReadAllLines(logsPath);

Assert.IsTrue(logs.Length > 0);

Assert.IsTrue(logs[logs.Length - 1].Contains("Invalid amount value"));

}

[TestMethod]

public void ValidateDate\_ThrowsExceptionForInvalidFile()

{

// Step 3: Provide an invalid file path and invoke the method

try

{

var result = excelProcessor.ValidateDate(invalidFilePath);

}

catch (Exception ex)

{

// Assert that an error message was thrown

StringAssert.Contains(ex.Message, "Error opening file nonexistent.csv:");

return;

}

Assert.Fail("Expected exception was not thrown.");

}

}