Lab 1

Here’s a small Python program (“Debug\_me.py”) designed as a **debugging learning exercise** in **Spyder IDE**. It includes intentional errors such as:

1. **Column name error**
2. **Normalization (scaling) problem**
3. **Wrong version of scikit-learn** (function used that might not exist in older versions)
4. **General poor coding practice to encourage step-by-step debugging**

**Instructions for Use**

1. Run the script in **Spyder IDE**.
2. Use **Spyder’s variable explorer and debugger** to step through and **identify the issues**.
3. Fix each issue iteratively using breakpoints and variable inspection.

**What Learners Should Find and Fix**

A screenshot of a chat

AI-generated content may be incorrect.

**Learning Outcomes**

* Use the **Spyder debugger to step through lines** and **inspect variables**
* Learn how to read **tracebacks** and **locate bugs**
* Fix issues iteratively using **variable explorer**, **console testing**, and **watch expressions**
* Understand importance of **version compatibility**

\*\*

Lab 2

Here's a **more challenging Python program** full of subtle bugs, designed to help learners **master debugging with Spyder IDE**.

**Objectives**

* Introduce **logic bugs**, **data leakage**, **NaN issues**, and **label misalignment**
* Reinforce debugging tools: **breakpoints**, **variable explorer**, **step into/over**, **watch window**
* Encourage inspection of intermediate outputs for **unexpected behaviours**

Scenario: Titanic Survival Prediction (with Bugs)

**Issues Learners Must Discover & Fix**

A screenshot of a computer

AI-generated content may be incorrect.

**What to Explore with Spyder**

* Set **breakpoints** inside preprocess() to check intermediate values.
* Use the **Variable Explorer** to compare df, encoded\_df, and X.
* Add **Watch expressions** for df.isnull().sum(), X.shape, and y.shape.
* Use the **step into/over** debugger to trace the flow of logic.
* Catch and fix the **Silent Bugs** (no traceback, but wrong predictions or accuracy).

**Extension Tasks**

* Fix all bugs and retrain the model correctly.
* Add accuracy\_score() to evaluate precision.
* Plot a bar chart of prediction outcomes using matplotlib.

\*\*

Lab 3

Implementing **safeguards and logging** in **Spyder** can significantly improve your debugging workflow and help you track down errors more systematically—especially when you’re working on data science or machine learning projects.

Below is a **step-by-step guide** to implement both **safeguards** (using assertions and exception handling) and **logging** in Spyder.

**PART 1: SAFEGUARDS (Assertions + Try/Except)**

**✅ Step 1: Open your script in Spyder**

Spyder uses an IPython Console and Variable Explorer, so you can interactively test your code. Create or open a .py script.

**✅ Step 2: Add Assertions to Catch Silent Errors**

**Use assert to check assumptions.**

**💡 Example:**

A close-up of a computer code

AI-generated content may be incorrect.

**Step 3: Wrap Risky Code in Try/Except Blocks**

**Use try/except to catch and handle unexpected errors gracefully.**

**💡 Example:**

A computer error message

AI-generated content may be incorrect.

**PART 2: IMPLEMENTING LOGGING IN SPYDER**

**✅ Step 1: Import and Configure the Logging Module**

Place this at the top of your script.

A computer screen shot of a computer code

AI-generated content may be incorrect.

**✅ Step 2: Replace print() with logging Calls**

Use the appropriate logging level:

A computer code with green text

AI-generated content may be incorrect.

A computer code with green and red text

AI-generated content may be incorrect.

Output is saved in debug.log, which you can open in Spyder’s **File Explorer pane**.

**Step 3: Run the Script and Check Logs**

1. Press **F5** or click **Run** in Spyder.
2. Go to the **File Explorer tab** and open debug.log.
3. You’ll see lines like:

A close-up of words

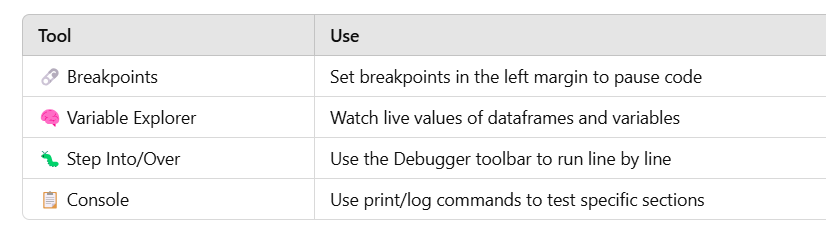
AI-generated content may be incorrect.

Step 4: Combine Logging + Try/Except for Full Protection

A screenshot of a computer program

AI-generated content may be incorrect.

BONUS: Debugging Tools in Spyder



\*\*

Lab 4

Here's the **corrected version of your original program** (‘fixed\_program.py’) that:

* ✅ Fixes all known bugs (column names, data access, and scaling)
* ✅ Is compatible with the unittest suite provided earlier
* ✅ Will now pass all tests if run together

Summary of Fixes

A screenshot of a computer program

AI-generated content may be incorrect.

A close-up of a test

AI-generated content may be incorrect.

\*\*

Lab 5

Below is a unittest suite that corresponds to your **intentionally faulty program (‘test\_faulty\_program’)**. These tests are designed to:

* Help identify the **bugs** step by step
* Be run inside **Spyder** or any terminal using python -m unittest
* Serve as a learning tool for debugging with **test failures as clues**

What These Tests Catch

A screenshot of a computer

AI-generated content may be incorrect.

**How to Run in Spyder**

1. Save this as test\_faulty\_program.py.
2. Press **F5** to run in Spyder.
3. Or run from the IPython Console with:

A black text on a white background

AI-generated content may be incorrect.

\*\*

Lab 6

**Step-by-Step: Set Up unittest for Python on GitHub**

**✅ Step 1: Organize Your Project Folder Locally**

Your project should follow a simple structure like this:

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 2: Write requirements.txt**

List your dependencies so GitHub Actions can install them.

A close-up of a white background

AI-generated content may be incorrect.

**Step 3: Add Your Test File**

Create a file named test\_fixed\_program.py in the root of your project. Include your unit tests (from earlier):

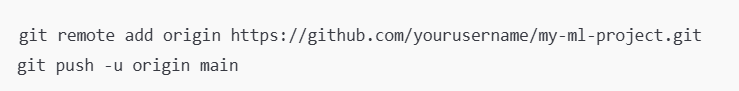
**Step 4: Push to GitHub**

1. Initialize a Git repo if you haven’t already:

A close-up of a computer screen

AI-generated content may be incorrect.

1. Push to a new GitHub repository:



**Step 5: Create GitHub Actions Workflow**

Create a folder .github/workflows/ and a file named python-tests.yml.

# .github/workflows/python-tests.yml

name: Python Unit Tests

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run tests

run: |

python -m unittest discover

Step 6: Commit and Push Workflow

A close-up of a computer code

AI-generated content may be incorrect.

**Step 7: Check GitHub Actions**

1. Go to your GitHub repo.
2. Click on the **"Actions"** tab.
3. You’ll see a workflow triggered.
4. Click to see logs for steps like “Install dependencies” and “Run tests”.

✅ If your tests pass, you’ll see a green check. If not, GitHub shows you the errors.

**BONUS: Badge for README**

Add this to the top of your README.md to show your test status:

![Python Tests](https://github.com/yourusername/my-ml-project/actions/workflows/python-tests.yml/badge.svg)

\*\*

Lab 7

**Download the zipped folder here**:  
my-ml-project.zip

**ncluded Files**

* fixed\_program.py: Your corrected machine learning script
* test\_fixed\_program.py: Unit tests using unittest
* requirements.txt: All required Python packages
* README.md: Project overview
* .github/workflows/python-tests.yml: GitHub Actions CI workflow

### Next Steps

1. **Unzip** the folder.
2. **Create a GitHub repo**, and push the contents:

A close-up of a computer screen

AI-generated content may be incorrect.

GitHub Actions will automatically run your tests after push! ✅

Provide screenshot that it is running (green tick) to be submited as part of the portfolio.