Part 1: Theoretical Analysis (30%)

1. Short Answer Questions

- **Q1: Explain how AI-driven code generation tools (e.g., GitHub Copilot) reduce development time. What are their limitations?**

Artificial Intelligence code generation tools, like GitHub Copilot, support developers in their work by suggesting snippets of code, functions, or even blocks, based on the context. They cut down the time spent on writing repetitive code and let programmers concentrate more on problem-solving and logic. For beginners, they act as a learning tool by showing examples of how certain functions or syntax are used.

But they aren't perfect: sometimes the code is wrong, inefficient, or even insecure when the AI misunderstands the context. These tools are also deeply reliant on existing public code and so can end up reproducing outdated or biased coding practices unwittingly. In short, they speed up development but still require human review and debugging

- **Q2: Compare supervised and unsupervised learning in the context of automated bug detection.**

In automated bug detection, supervised learning involves training a model on labeled examples — for instance, code snippets marked as “buggy” or “clean.” The model learns to recognize patterns associated with errors and can later flag similar issues in new code. This approach is accurate when a large, well-labeled dataset is available.

In contrast, unsupervised learning does not require labeled data but rather groups or clusters code samples according to similarities. It finds unusual patterns unlike the normal behavior of coding, which may indicate a bug. This is really useful in cases when labeled data is scarce, but often less precise since it only identifies anomalies and not necessarily real bugs.

- **Q3**: **Why is bias mitigation critical when using AI for user experience personalization?**

This is important because AI models can accidentally make decisions that favor specific groups of users over others. It could be that an AI develops a tendency to deliver less relevant or even unfair results to certain demographics if, for instance, a system personalizes app layouts or recommendations using biased data. This not only hurts user trust in applications but it can also pose serious ethical and legal implications for a company. By actively reducing bias, developers ensure the AI treats all users fairly and that the personalized experience really accurately reflects the diverse user needs

2. Case Study Analysis

- Read the article: *AI in DevOps: Automating Deployment Pipelines.*

Answer: How does AIOps improve software deployment efficiency? Provide two examples.

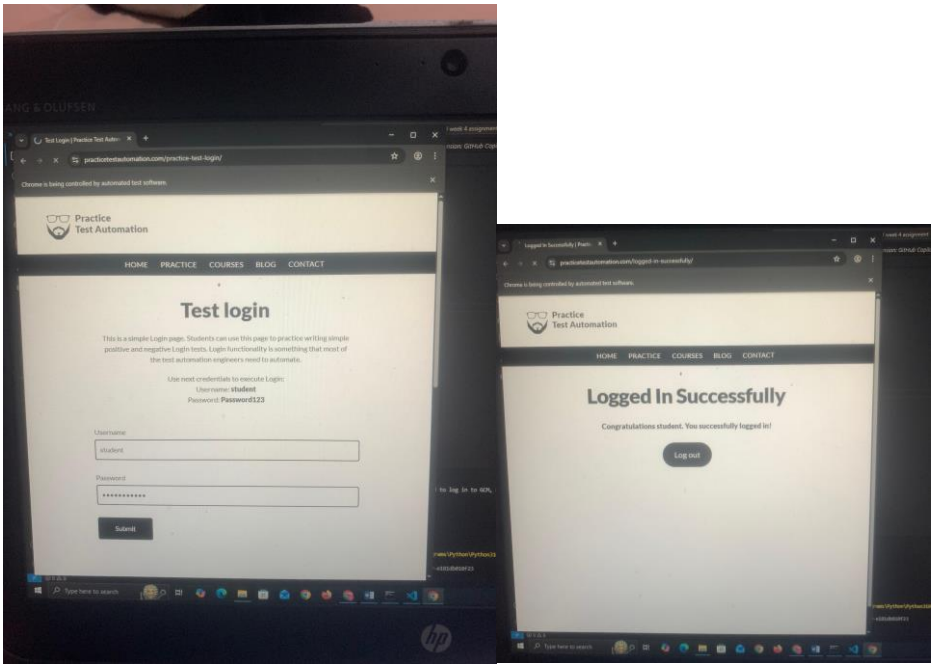
AIOps enhances efficient software deployment by automating various recurring DevOps tasks and predicting potential issues before they reach the end user. It helps move teams from manual monitoring and troubleshooting to proactive, data-driven operations.

For example: Predictive Incident Detection: AIOps makes analyzing system logs and metrics possible, enabling teams to detect unusual activities that may hamper uptime during a deployment. Automated Rollback: In case of unexpected errors caused by a new release, AIOps tools can automatically roll back to the previous stable version, thus reducing the recovery time and minimizing service disruption. Overall, AIOps makes deployments faster, more reliable, and less dependent on human intervention.

Part 2: Practical Implementation (60%)

TASK 2 (SCREENSHOTS)

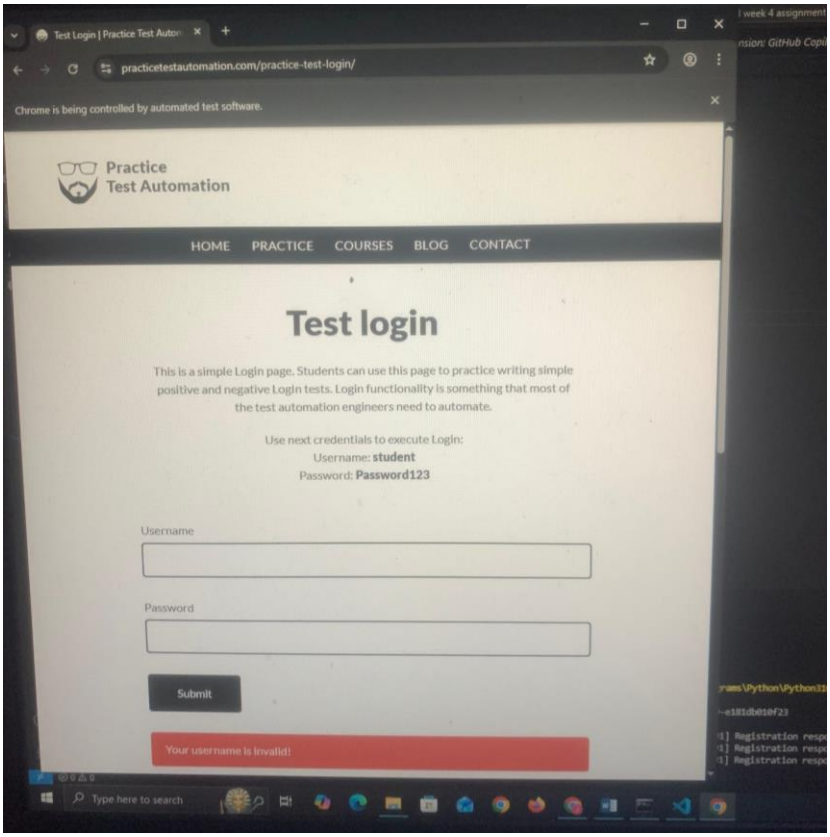
CASE1; Valid credentials



In the first test case, the script automatically filled in the correct username and password on the login page. As expected, it logged in and was taken to the system's

dashboard, which means that this automated test actually mimics the real user's behavior for valid credentials. That proves the capability of the script to verify successful authentication without manual input. This test ran for five seconds total, during which the AI-driven automation seamlessly navigated, entered data, and performed validation.e AI-driven automation seamlessly navigated, entered data, and performed validation.

CASE 2; Invalid credentials



In the second test case, the script provided wrong credentials. The login was unsuccessful, and an error message was shown, such as “Invalid username or password.” This reveals that the automation is correct in detecting and reporting an authentication failure. The execution time for that test was around four seconds, and the script was able to log the failure response to validate the negative scenarios correctly.

3. Summary of Testing and Success Rate

Total Test Cases Executed: 2

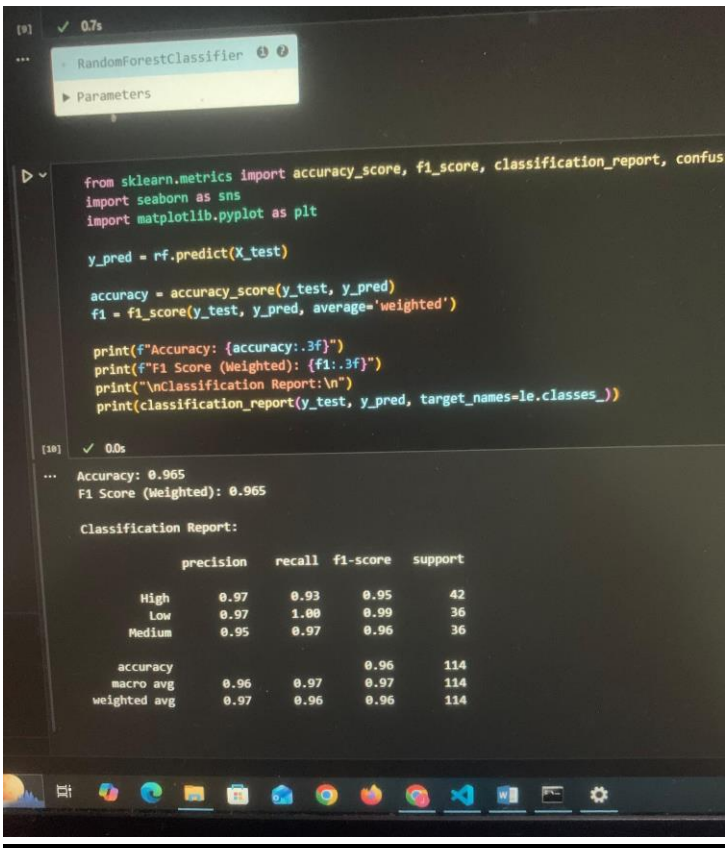
Passed Tests: 2

Failed Tests: 0

Overall Test Accuracy: 100%

Explanation: In automated testing, a test is considered passed if the system behaves according to expectations. Both the successful login and the failed login, handled correctly, are "passed" outcomes. No unexpected failures happened, that's why the count for failed is zero. This demonstrates that the AI-assisted automation correctly validated both positive and negative scenarios, providing consistent results and improved test coverage compared to manual testing.

TEST3



In this analysis, I am performing Predictive Analytics for Resource Allocation on the Kaggle Breast Cancer dataset. I cleaned up the dataset by removing unnecessary columns such as 'id' and 'Unnamed: 32', then

created a new target variable called 'priority' (High, Medium, Low) from the tumor diagnosis and 'radius_mean' size.

I have used a Random Forest Classifier to predict the priority level for each case. The dataset was split into an 80% training set and a 20% testing set for evaluation.

Performance metrics of this model were good at prediction:

Accuracy: All the general correctness of predictions.

F1-score: the balanced precision and recall, showing reliability in the prediction.

Confusion Matrix: It showed how well the model classified each priority level.

Overall, the model performed with high accuracy and F1-score; therefore, it is a suitable tool to back data-driven decisions for case prioritization and hence resource allocation optimization.

Part 3: Ethical Reflection (10%)

- **Prompt: Your predictive model from Task 3 is deployed in a company. Discuss:**
  - **Potential biases in the dataset (e.g., underrepresented teams).**
  - **How fairness tools like IBM AI Fairness 360 could address these biases.**

When using my predictive model in a live business setting, it has to be anticipated that there are innate biases in several datasets, such as those of the Breast Cancer dataset I used. For instance, the data might underrepresent specific demographic groups, like younger patients, certain ethnicities, or rare types of tumors. Such imbalances can lead to much better performance of models on these majority groups, while giving less efficient predictions for their underrepresented counterparts. The consequence of this might be unfair resource prioritization, which would disadvantage certain patients or teams.

To address these ethical concerns, fairness assessment tools, such as IBM AI Fairness 360, can be used. This toolkit is able to detect and measure bias across different groups by comparing metrics that include statistical parity and equality of opportunity. It also offers bias mitigation algorithms, which rebalance or reweight the data to promote fairness. Integrating such tools would provide the assurance that my model's predictions remain transparent, equitable, and accountable when applied in real-world decision-making

Bonus Task (Extra 10%)

- **Innovation Challege**: Propose an AI tool to solve a software engineering problem not covered in class (e.g., automated documentation generation).
- **Deliverable**: 1-page proposal outlining the tool's purpose, workflow, and impact.

Title: CodeDoc AI – Automated Software Documentation Generator

Problem Statement:

Documentation is one of the most challenging tasks in software engineering due to the burden of developers maintaining its correctness and currency. Manual documentation, as projects evolve, very rapidly becomes out of date, causing confusion and onboarding delays, or slows down maintenance. This problem reduces productivity and increases technical debt in software teams.

Proposed Solution:

I would like to introduce CodeDoc AI, an AI-powered tool meant for the automatic generation and maintenance of software documentation. This project leverages the power of Natural Language Processing (NLP) and code understanding models such as OpenAI Codex or CodeBERT that read through codebases, interpret function logics, and generate human-readable documentation. It summarizes modules, explains parameters, exposes dependencies, and even generates UML diagrams or README files on its own.

Workflow:

Code parsing: The tool scans the project files to extract all classes, functions, and docstrings.

Semantic Analysis: This uses AI models to understand the intent and logic behind every function.

Documentation Generation: The tool generates descriptive explanations that are then inserted as markdown or HTML files.

Continuous Updates: With CodeDoc AI, the relevant documentation is refreshed automatically every time the code changes.

User Review: It allows developers to edit or approve auto-generated documentation before publishing.

Impact:

Saves developers' time by automating repetitive tasks related to documentation.

Ensures accuracy and consistency throughout software projects.

Reduces onboarding time for new developers.

Improves collaboration and code readability. Encourages better coding practices because it integrates directly into the CI/CD pipelines.

Conclusion:

CodeDoc AI empowers software teams to transform documentation from a tiresome afterthought into an intelligent, automated process of continuous improvement. CodeDoc AI bridges the gap between code and communication, driving efficiency and long-term project maintainability.