Name:- Abemelek Samson Abduke

Week 4 Assignment: AI in Software Engineering Report

Part 1: Theoretical Analysis

1. Short Answer Questions

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

AI-powered code generation tools like GitHub Copilot save developers a lot of time by suggesting code snippets or even entire functions while you type. Instead of writing repetitive boilerplate code, you can focus on solving the actual problem. They also help with learning new libraries and keeping code consistent across a project.

However, they aren't perfect. Sometimes the generated code can have bugs or security issues, and for more complex logic, you still need to refine it manually. Over-relying on these tools can also make developers less confident in problem solving, and there's sometimes confusion about the copyright or licensing of generated code.

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

Supervised learning in bug detection uses labeled datasets where each example is marked as a bug or not. The model learns these patterns to automatically identify similar bugs in the future. This works well if you have a lot of historical bug data, but labeling data can be time consuming.

Unsupervised learning, on the other hand, doesn't need labeled data. It looks for anomalies or unusual patterns in code that might indicate bugs. This is great for finding new or unexpected bugs, but it can sometimes flag normal behavior as a bug, so developers need to review results carefully.

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

When AI personalizes user experiences, it learns from historical data. If that data is biased, say, favoring one group over another, the AI can reinforce those biases, giving unfair or frustrating experiences to some users. By mitigating bias, we make sure AI recommendations are fair and inclusive, improving satisfaction for everyone and building trust in the system.

2. Case Study Analysis: AIOps in Software Deployment

AIOps, which combines Artificial Intelligence and Machine Learning with DevOps, has transformed the way software is deployed by automating critical tasks, enabling smarter decisions, and reducing risks. By embedding AI into deployment pipelines, teams can focus less on repetitive manual work and more on innovation and quality.

How AIOps Improves Software Deployment Efficiency:

1. **Automating Repetitive Tasks and Accelerating Cycles:** AI automates many manual steps in the development and deployment process. This makes deployment cycles much shorter and workflows smoother. Essentially, AI allows repetitive tasks to happen behind the scenes, freeing developers to focus on complex problem solving while the system handles routine operations.
2. **Predicting and Preventing Failures:** AI can analyze historical data and detect patterns that may indicate potential build or installation failures. By predicting these problems before they occur, AIOps ensures that deployments are less likely to fail when going live. This proactive approach reduces downtime and increases overall system reliability.
3. **Minimizing Human Intervention and Error:** With AI handling more of the deployment workflow, the need for human intervention decreases. This not only speeds up the rollout process but also reduces operational mistakes, configuration errors, and other human induced risks, making deployments more robust and dependable.
4. **Optimizing Testing Workflows:** AI can analyze past test outcomes to determine which tests are most likely to succeed or fail. This allows CI/CD workflows to prioritize high-impact tests first, giving developers faster feedback and enabling more efficient iterations. The result is a smoother deployment process with fewer delays caused by test bottlenecks.

Examples of AIOps in Action:

- **Harness:** This platform uses AI to automatically roll back failed deployments. By doing so, it minimizes the need for human intervention and reduces downtime, allowing teams to maintain continuous and reliable software delivery.
- **Facebook:** The company leverages AI powered test automation to decrease errors introduced during deployment. Its AI system predicts which tests might "flake out" and prioritizes them accordingly, making deployments faster and more reliable.
- **Netflix:** Netflix employs AI to automate canary deployments and monitor performance issues before they impact users. This ensures highly controlled rollouts and uninterrupted viewing experiences, demonstrating how AI can enhance both efficiency and user satisfaction.

Overall, AIOps not only speeds up deployment but also makes the process smarter, safer, and more reliable by combining predictive insights with automation. Companies adopting these AI driven approaches can achieve faster, smoother, and less error-prone software releases.

Part 2: Practical Implementation Summaries

Task 1: AI-Powered Code Completion Analysis

The **manual version** is simpler and easier to read, but it does not handle missing keys as elegantly as the Gemini AI version. If a dictionary lacks the specified key, the manual version either ignores it or prints a warning, while the AI version automatically places it at the end of the list.

The AI-generated function uses a helper function to return a tuple `(found_flag, value)` for each dictionary, ensuring that missing keys don't break the sorting and maintaining a stable order. This makes it safer for production, especially when datasets are inconsistent.

In terms of **efficiency**, both versions rely on Python's `sorted()` function with O(n log n) time complexity, so performance is similar. The main difference is in **robustness and maintainability**: the Gemini version is more foolproof and requires less manual intervention for edge cases, while the manual version is more transparent and easier to understand for beginners.

Overall, combining AI suggestions with a manual check is ideal: developers can quickly generate reliable code using AI while reviewing and adjusting it to match project-specific needs.

Task 2: Automated Testing with AI Summary

Objective:

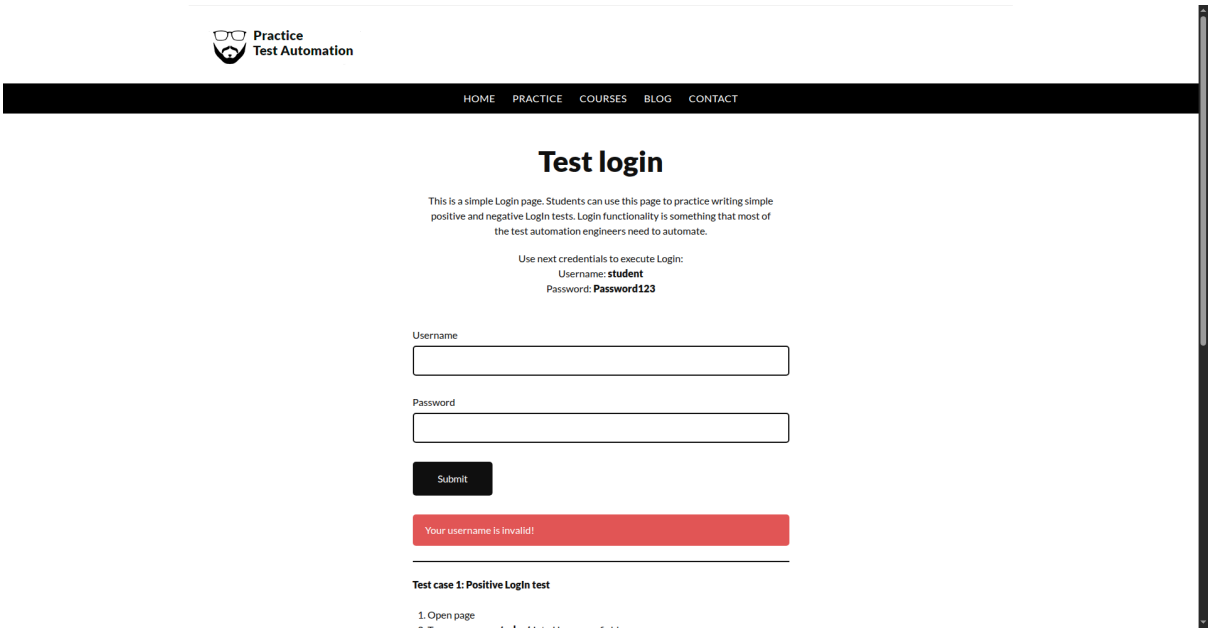
The goal of this task was to automate a login functionality test using Selenium WebDriver. The script verifies both valid and invalid login attempts on a sample web page.

Steps Performed:

1. Installed and configured Selenium with Chrome WebDriver.
2. Launched the browser and navigated to the target login page.
3. Located username and password input fields using By.ID.
4. Performed two test cases:
    a. Valid Login Test: Entered correct credentials and checked for a success message.
    b. Invalid Login Test: Entered incorrect credentials and verified the error response.
5. Used WebDriverWait to ensure elements loaded properly.
6. Captured a screenshot after test execution and closed the browser.
7. Results:
    a. Valid Login Test: PASSED
    b. Invalid Login Test: PASSED
8. Screenshot successfully saved after execution.

Conclusion:

The test automation works correctly and reliably detects both valid and invalid login attempts. This demonstrates the use of Selenium for automated web application testing.



Task 3: Predictive Analytics for Resource Allocation Summary

Objective:

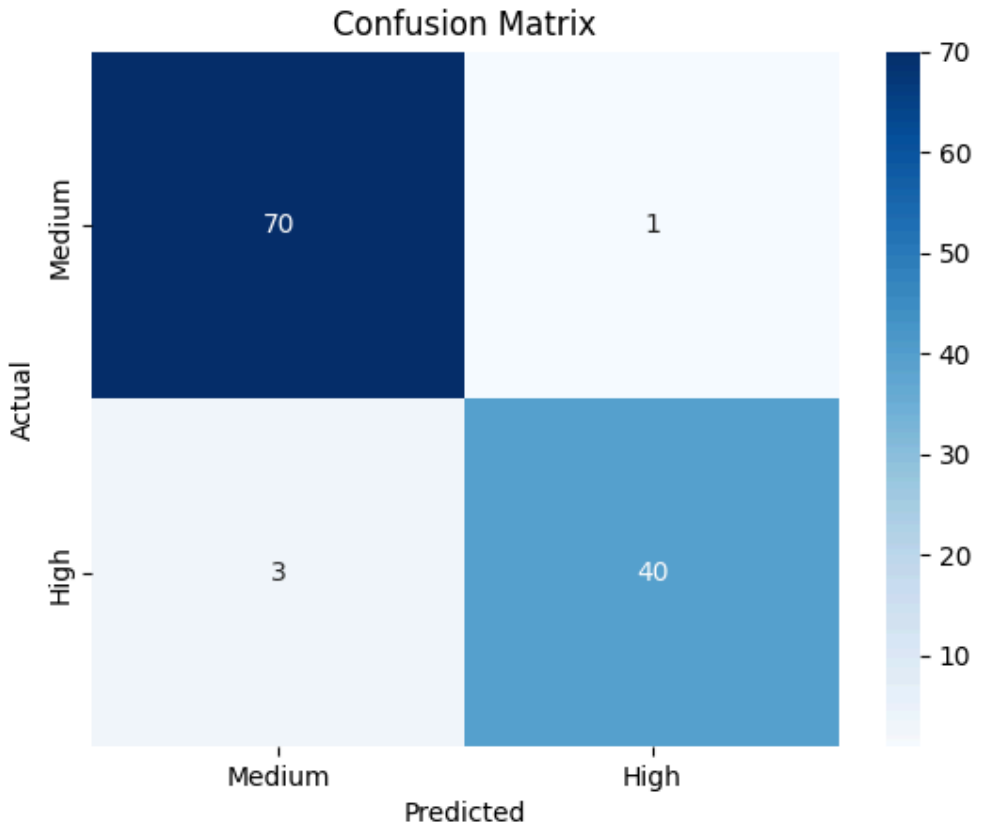
The goal of this task was to apply predictive analytics to simulate issue priority classification using the Kaggle Breast Cancer dataset. The task demonstrates how AI models can aid in resource allocation by predicting which issues require high or medium attention.

Methodology:

1. Data Preprocessing: The dataset was loaded, and unnecessary columns such as id were removed. The target variable, originally diagnosis (Malignant/Benign), was mapped to numerical labels representing issue priority: high (Malignant) and medium (Benign).
2. Feature Selection & Splitting: All relevant features were retained for training. The dataset was split into training (80%) and testing (20%) subsets.
3. Model Training: A Random Forest Classifier was trained on the training data to predict issue priority.
4. Evaluation: The model was evaluated using accuracy and F1-score, with additional insights from a confusion matrix.
5. Results:
    a. Accuracy: 96.5%
    b. F1-score: 96.5%

Conclusion:

The predictive model performed excellently in classifying issue priority. This task illustrates how AI can enhance decision-making in software engineering by predicting critical issues, enabling teams to allocate resources effectively. The confusion matrix and performance metrics demonstrate the model's reliability and applicability to real-world scenarios.



Part 3: Ethical Reflection

Deploying a predictive model for issue prioritization in a company raises several ethical considerations. One key concern is bias in the dataset. For example, if certain types of issues or teams are underrepresented in the training data, the model may systematically underestimate or overestimate their priority. This could lead to unfair allocation of resources, neglecting critical issues or over-focusing on less important ones.

To mitigate these biases, fairness tools like IBM AI Fairness 360 can be applied. These tools detect and quantify bias in both features and predictions, enabling developers to reweight or adjust the dataset to ensure equitable outcomes. Techniques such as resampling, equalized odds, or adversarial debiasing can help create a model that treats all issue types and teams fairly.

Finally, transparency and explainability are critical. Stakeholders should understand how the model makes predictions, allowing human oversight to intervene when necessary. Overall, incorporating fairness and ethical considerations ensures that AI-driven resource allocation improves efficiency without compromising equity or accountability.