Part 1: Theoretical Analysis

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?

  The tools reduce development time by: Providing instant code suggestions and auto-completion, generating boilerplate code and common patterns, offering solutions for repetitive tasks, reducing context switching between documentation and coding, and helping with API integration and library usage. All these together reduce keystrokes and cognitive load and offer effective solutions for routine tasks and for accelerating prototypes.

  Limitations: May generate insecure or inefficient code, limited understanding of business context, potential copyright issues with training data, can produce code that passes tests but doesn't meet requirements, and over-reliance may reduce developer learning.

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

  Supervised learning requires labeled examples (buggy vs. non-buggy) and excels when historical bug labels exist; classifiers can predict whether new code changes are likely buggy. Unsupervised learning finds anomalies without labels, useful for detecting novel or rare failure modes by flagging deviations in logs, metrics, or test outcomes. Supervised methods give precise predictions when labels are available; unsupervised methods offer broader discovery for unknown issues.

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

  Personalization models amplify historical patterns. If training data underrepresents groups or encodes unfair preferences, models will systematically privilege some users and degrade others' experiences. Bias mitigation preserves fairness, avoids legal/ethical harms, and maintains trust. Practically, it improves product reach and reduces the risk of discriminatory outcomes that can harm users and the company, i.e., essential for ethical AI implementation

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 improves deployment efficiency by predicting failures and prioritizing actions, and by automating monitoring and remediation. **Intelligent Failure Prediction:** AI analyzes historical deployment data to predict potential failures, allowing proactive mitigation. **Automated Rollback Decisions**: Machine learning models can automatically detect deployment issues and trigger rollbacks and suggest remediation steps, cutting mean time to recovery faster than human operators.

Part 2: Practical Implementation (60%)

Task 1: AI-Powered Code Completion

Comparison and efficiency between the AI-suggested code with manual implementation.

Both implementations use Python's built-in sorted() function with lambda, demonstrating efficient sorting. The primary difference is how missing keys are handled. The manual version explicitly groups missing keys and preserves stability and deterministically places missing values at the end. The manual version directly accesses the dictionary key, which could raise KeyError if the key is missing. The AI-suggested version uses dict.get() with a default value of 0, providing better error handling.

The AI version is more robust for production environments where data might be inconsistent. However, the default value of 0 might not always be appropriate - for string keys, this could cause TypeErrors. The manual implementation is more transparent but less safe.

In terms of efficiency, both have O(n log n) time complexity. The AI version has slightly more overhead due to method calls but offers better reliability. For this specific use case, a hybrid approach would be ideal: using .get() with appropriate default values based on expected data types.

The AI tool demonstrated understanding of common edge cases and provided a more production-ready solution, though it required domain knowledge to validate the appropriateness of the default value.

Task 2: Automated Testing with AI: Framework: Use Selenium IDE with AI plugins or Testim.io.

Explain how AI improves test coverage compared to manual testing.

AI improves test coverage by generating test variations, suggesting edge cases, and reducing flaky tests via smart assertion and retry strategies. Tools like Testim or AI-enhanced Selenium wrappers analyze DOM changes and recommend robust selectors, reducing maintenance. Automated test generation increases the breadth of scenarios covered per commit and focuses human testers on exploratory and high-value testing.

Task 3: Predictive Analytics for Resource Allocation

The Jupyter notebook contains the script, while the screenshot folder has performance metrics (accuracy and F1-score).

Accuracy: 0.9649

F1-Score: 0.9653

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.91	0.95	64
1	0.99	1.00	0.99	79
2	0.85	1.00	0.92	28
accuracy			0.96	171
macro avg	0.95	0.97	0.95	171
weighted avg	0.97	0.96	0.97	171

Part 3: Ethical Reflection (10%).

Answers are in the ethical reflection markdown.

Bonus Task (Extra 10%)

Answers are in the bonus task markdown.