✅ Part 1: Theoretical Analysis (30%)

Q1: AI-Driven Code Generation Tools

Answer:
AI code generation tools like **GitHub Copilot** reduce development time by suggesting code snippets in real-time, automating boilerplate writing, and even generating entire functions from comments. Developers save time on syntax, API calls, and common algorithms, allowing them to focus more on logic and design.

Limitations:

- May generate incorrect or insecure code.
- Lacks deep contextual understanding.
- May replicate biased or low-quality training data.
- Requires constant human oversight and validation.

Q2: Supervised vs Unsupervised Learning in Bug Detection

Aspect	Supervised Learning	Unsupervised Learning
Approach	Uses labeled bug data (e.g., bug/no-bug).	Detects anomalies or clusters without labels.
Use Case	Classify known bug types.	Discover unusual behavior (potential bugs).
Limitation	Needs large labeled datasets.	May flag false positives.

Example:

- **Supervised**: Random Forest classifies crash logs.
- **Unsupervised**: K-means clusters abnormal system behavior.

Q3: Importance of Bias Mitigation in Personalization

Bias in AI-driven personalization leads to:

- **Discrimination** (e.g., biased content recommendations).
- **Exclusion** of user groups.
- **Echo chambers**, reinforcing stereotypes.

Bias mitigation ensures fairness, diversity, and inclusivity. It prevents algorithms from favoring one demographic over another and supports ethical and user-centric software design.

Case Study: AIOps in DevOps

How AIOps improves deployment:

1. **Anomaly detection**: AI identifies unusual behavior during deployment (e.g., error spikes).
2. **Automated rollback**: Machine learning models detect failures and trigger recovery scripts.

Examples:

- AI auto-scaling cloud instances based on traffic predictions.
- AI-powered root cause analysis reducing debugging time.

✅ Part 2: Practical Implementation (60%)

Task 1: AI-Powered Code Completion - (Naomi Wairimu and Faith Wafula)

Tool Used: GitHub Copilot
Task: Sort list of dictionaries by a key (e.g., "age")

Analysis (200 words):
GitHub Copilot produced a concise version with `.get()` for fault tolerance. It's practical in scenarios where keys might be missing. My manual implementation uses `sorted()`, which is more functional and doesn't mutate the original list. Copilot's solution is slightly more efficient due to in-place sorting but risks unintended side effects. In mission-critical systems, immutability might be preferred.

Task 2: Automated Testing with AI - (Obondo Patrick, Pharix Eloga)

Tool Used: Selenium IDE + Testim.io
Task: Login test automation

- Valid login: ✅
- Invalid login: ✅ (error messages validated)

Screenshot: *(Include Selenium IDE/Testim result)*
Summary (150 words):
AI-enhanced tools like Testim use visual recognition and DOM analysis to reduce test flakiness. AI adapts to UI changes, unlike brittle manual XPath selectors. This leads to higher **test coverage**, better **test stability**, and lower **maintenance costs**. It also suggests new test cases by analyzing user flows and behaviors.

Task 3: Predictive Analytics – Resource Allocation - (Jeremiah Katumo)

Dataset: Kaggle Breast Cancer Dataset
Goal: Predict priority level (High/Medium/Low) based on features

Workflow:

1. Preprocessing: Handle missing values, encode labels.
2. Model: `RandomForestClassifier()`
3. Evaluation:

Notebook Deliverable: `breast_cancer_priority_prediction.ipynb`
Results:

- Accuracy: 96.5%

- F1 Score: 0.96

✅ Part 3: Ethical Reflection (10%)

Reflection:

Bias Risk:
If the dataset overrepresents one team or location (e.g., majority US-based reports), the model might predict “High Priority” inaccurately for underrepresented teams.

Fairness Solutions:
Using tools like **IBM AI Fairness 360**, we can:

- Evaluate disparate impact metrics.
- Rebalance datasets.
- Use algorithms like reweighing or bias mitigation during training.

Outcome:
Improved fairness, transparent predictions, and trust in the system.

🏆 Bonus Task: Innovation Challenge (10%)

Tool Proposal: DocuGenAI – AI-Powered Software Documentation Generator

Problem:

Developers struggle with maintaining up-to-date documentation.

Solution Workflow:

1. Extracts code comments + structure using AST.
2. Uses NLP (e.g., T5 or GPT) to generate technical docs.
3. Syncs with GitHub repos and highlights undocumented changes.

Impact:

- Reduces manual documentation time.
- Boosts onboarding and code comprehension.
- Ensures consistent API and logic explanations.

Tech Stack:

- Python, TreeSitter, HuggingFace Transformers.
- GitHub Webhooks.
- Optional VS Code plugin.

✅ Submission Checklist:

Deliverable	Format	Shared On
✅ Report	PDF (with answers, screenshots, reflections)	Community as article
✅ Code	Python/Selenium Notebook/Scripts	GitHub repo
✅ Video Demo	~3 minutes	Group post for peer review