Assignment Report ~ Theoretical Analysis.

Q1. How AI-driven code generation tools reduce development time, their limitations

AI code completion tools reduce developer time by suggesting boilerplate, standard idioms, and one-line solutions that would otherwise take manual typing and lookups. They speed prototyping, suggest unit tests, and lower cognitive load for repetitive code. Formally, if $T_m$is manual time and $T_a$is time with assistance, typical savings arise because $T_a = T_m - S$where Scaptures search/typing efficiency and reuse of learned patterns.

Limitations include hallucination (incorrect or insecure suggestions), license/attribution issues (copied patterns from training data), over-reliance that degrades deep understanding, and brittle behaviour for domain-specific logic. They may not encode project-specific context (architecture, hidden constraints). Human review is mandatory.

Q2. Supervised vs unsupervised learning in automated bug detection

Supervised learning uses labeled examples (bug vs non-bug code). It trains a classifier $f(x)$to predict labels, enabling precise detection when labeled data covers patterns. Unsupervised learning finds anomalies or clusters in unlabeled data: it surfaces unusual code changes or behaviors (e.g., outlier static analysis scores) without explicit labels. Supervised gives higher precision when labels exist; unsupervised helps discover novel bugs and scale to unlabeled repositories.

Q3. Why bias mitigation is critical for UX personalization

Personalization models shape what users see; biased models amplify skewed exposures and can exclude or harm groups (e.g., under-served locales, accessibility needs). Fairness is needed to avoid reinforcing stereotypes or unequal access. Mitigation ensures equitable treatment metrics and preserves trust and regulatory compliance.

Case Study: *AI in DevOps: Automating Deployment Pipelines* — How AIOps improves deployment efficiency (two examples)

AIOps improves deployment by:

(1) **Automated anomaly detection and rollback** — monitoring models detect abnormal metrics (latency, error rate) and trigger automated rollbacks or canary traffic reductions, reducing mean time to recovery.

(2) **Intelligent pipeline optimization** — AIOps can predict flaky tests and reorder/parallelize test suites to reduce feedback time, and suggest minimal commit sets for rebuilds, cutting CI/CD cycle time.

Practical Implementation:

Task 1:

The AI-suggested version is concise, and leverages Python's sorted with a simple get fallback. It is efficient: it uses Timsort (O (n log n)) and minimal overhead. However, it assumes comparability of values and silently places missing keys as None, which may raise TypeError when mixing incomparable types (e.g., int and str) or produce unintended ordering.

The manual implementation prioritizes robustness and clarity. It validates input types, distinguishes missing keys using a tuple (exists flag, value) so missing keys uniformly sort last, and includes a fallback to string conversion when comparisons fail. This adds small constant-time overhead in key construction but avoids runtime exceptions and ambiguous ordering.

In most typical datasets (homogeneous comparable key values), both versions are equivalent in asymptotic cost; the AI version is slightly faster due to fewer checks. In real-world, heterogeneous or messy data is common; the manual implementation yields more reliable and maintainable behaviour. For production code, prefer the manual approach (explicit checks and documented fallbacks). For rapid prototyping where data cleanliness is guaranteed, the AI suggestion is acceptable.

Task 2:

AI-powered testing tools like Testim.io or Selenium with AI plugins can analyse past test results to detect flaky steps, auto-heal element locators, and generate additional test cases for edge conditions. In this example, Selenium automates login validation, reducing manual effort and ensuring consistency. With AI assistance, such tests can automatically adapt when the HTML structure changes (e.g., AI finds the correct “Login” button) or when new input combinations appear. Compared to manual testing, AI-driven automation improves coverage by exploring diverse input patterns, recognizing test failure causes (network vs logic errors), and prioritizing high-risk areas. This leads to more robust test suites, faster releases, and fewer production defects.

Task 3:

Accuracy: 0.9736842105263158

F1 (macro): 0.9719582489484343

	precision	recall	f1-score	support
low	1.00	1.00	1.00	24
medium	0.92	0.96	0.94	24
high	0.98	0.97	0.98	66
accuracy			0.97	114
macro avg	0.97	0.98	0.97	114
weighted avg	0.97	0.97	0.97	114

Confusion matrix:

[[24 0 0]

[0 23 1]

[0 2 64]]

The Random Forest model demonstrated excellent predictive performance, achieving an overall accuracy of 97.4% and a macro F1-score of 0.97. This indicates that the model is highly reliable in classifying issue priorities into low, medium, and high categories. The per-class analysis shows perfect performance for low-priority issues (F1 = 1.00), strong results for high-priority issues (F1 = 0.98), and slightly lower but still robust performance for medium-priority issues (F1 = 0.94). The confusion matrix confirms that only three cases were misclassified, with minimal overlap between medium and high priorities. These results suggest that the model effectively captures key patterns in the data, enabling accurate and consistent predictions. Overall, the model can be confidently used for predictive analytics in software resource allocation, allowing teams to prioritize tasks efficiently and reduce delays in addressing high-impact issues.

Ethical Reflection

If the predictive model is deployed to assign *issue priorities* across engineering teams:

Potential biases:

- Dataset mismatch: using a biomedical dataset as proxy causes domain mismatch. In real deployment, the model trained on historical issue data could encode organizational bias: teams with fewer reporters or lower visibility may have fewer “high” labels and be under-prioritized.
- Label bias: human triage decisions can have historical bias (priority labels reflect past manager preferences).
- Feature bias: features correlated with team membership (e.g., module name) can proxy for demographics or resource allocation.

How IBM AI Fairness 360 could help:

- Use fairness metrics (disparate impact, equal opportunity) to detect whether priority assignment disproportionately harms teams or groups.
- Apply mitigation techniques: re-weighting, adversarial debiasing, or post-processing (threshold adjustments) to ensure parity across protected groups or teams.
- Run pre-deployment dashboards to visualize fairness trade-offs and to present clear audits to stakeholders.

Bonus Task — Innovation Challenge

Tool name: DocGen-AI: Contextual, Living Documentation Generator for Repos

Purpose: Automatically generate, update, and maintain developer-facing documentation (API docs, design rationales, README sections, and usage examples) linked directly to source code and test coverage so docs never drift.

Workflow

1. **Repo Scanner** — periodic scan of the repository (commits, PRs) and test coverage reports.
2. **Context Extraction** — static analysis to extract function signatures, types, docstrings; runtime traces from tests to gather usage examples.
3. **Natural-Language Generation** — produce or update documentation sections using an LLM with local code context and changelog; generate runnable examples from tests.
4. **Diff & PR Generator** — propose doc changes as a draft PR with side-by-side diffs; CI runs smoke checks ensuring code snippets compile/run.
5. **Human-in-the-loop Review** — reviewers accept/modify the PR comment which improves the model's future suggestions (reinforcement from edits).
6. **Living Links** — docs include live badges showing last verified commit and test coverage, with deep links to the code lines.

Impact

- Reduces time spent writing/maintaining docs.
- Keeps documentation aligned with code and tests, improving onboarding and reducing confusion.
- Encourages higher test coverage (since examples come from tests).
- Minimizes outdated docs which cause security and maintenance issues.