Week 4 Assignment: AI in Software Engineering

Theme: Building Intelligent Software Solutions

Name: Sophia Nakhan

Part 1: Theoretical Analysis (30%)

Q1: How AI-Driven Code Generation Tools Reduce Development Time + Limitations

AI-driven code generation tools like GitHub Copilot significantly reduce development time by providing real-time code suggestions, completing boilerplate code, and even generating entire functions based on comments or partial inputs. They enhance productivity by automating repetitive tasks, allowing developers to focus more on logic, architecture, and creativity.

However, their limitations include:
- Context awareness: They may generate incorrect or suboptimal code if the context is not well-defined.
- Security issues: Generated code might contain vulnerabilities.
- Over-reliance: Developers may stop learning or understanding the code if they depend too heavily on suggestions.
- Bias: The training data may reflect outdated or biased coding practices.

Q2: Compare Supervised vs. Unsupervised Learning in Automated Bug Detection

In automated bug detection:
- Supervised learning uses labeled data (e.g., bug reports labeled as "bug," "enhancement," or "feature request") to train models that can classify new issues. It provides high accuracy when labels are available, making it ideal for large, labeled datasets.
- Unsupervised learning identifies patterns or anomalies in unlabeled data. It can detect unusual code behavior or cluster similar bug reports, useful when labels are missing or incomplete.

Summary: Supervised learning excels in structured environments, while unsupervised learning is better for discovering hidden structures or anomalies.

Q3: Why Bias Mitigation Matters in AI for User Experience Personalization

Bias mitigation is critical in AI-driven user experience personalization because biased algorithms can lead to exclusion, stereotyping, or unfair treatment of certain user groups. For example, a recommendation system trained on biased data might favor content suited

only to majority users, ignoring minority needs.

This harms inclusivity and user satisfaction. Techniques like balanced datasets, fairness-aware algorithms, and auditing tools ensure that AI systems treat all users equitably and adapt to diverse preferences.

Case Study: How AIOps Improves Deployment Efficiency + 2 Examples

AIOps enhances software deployment efficiency by using machine learning and data analytics to automate monitoring, detect anomalies, and predict failures. It reduces manual effort, accelerates deployment cycles, and improves reliability.

Examples:
1. Anomaly Detection in CI/CD Pipelines: AIOps tools like Moogsoft detect unusual build failures or latency, enabling faster rollback or remediation.
2. Predictive Scaling: Platforms like Dynatrace use AI to predict workload spikes and auto-scale infrastructure, preventing crashes during deployments.

Part 2: Practical Implementation (60%)

Task 1: AI-Powered Code Completion

Manual Implementation:

```
def sort_by_key(data_list, sort_key):
    return sorted(data_list, key=lambda x: x[sort_key])

# Sample data
data = [
    {"name": "Alice", "age": 20},
    {"name": "Bob", "age": 30},
    {"name": "Charlie", "age": 22}
]

# Sort by age
sorted_data = sort_by_key(data, "age")
print("Manual Sorted:", sorted_data)
```

AI (Gemini) Suggested Implementation:

```
def sort_by_key(data_list, sort_key):
    return sorted(data_list, key=lambda x: x[sort_key])
```

Reflection:
In this task, I created a Python function to sort a list of dictionaries by a specific key, using both a manual approach and an AI-generated suggestion from Gemini AI. My manual implementation used the `sorted ()` function with a lambda, which is a well-known and efficient method in Python.

After prompting Gemini with a request to write the same function, it recognized that I had already written one and generated nearly identical code. This showed how AI tools are capable of understanding the existing context and offering clean, efficient code suggestions based on common patterns.

There was no significant performance difference between the two versions, but using Gemini AI saved time and confirmed best practices. The experience highlighted how AI tools can accelerate repetitive or boilerplate tasks, making the development workflow more efficient.

However, it also emphasized the importance of developer oversight. Even though the code was accurate in this case, AI tools can sometimes produce incorrect logic if used without understanding or testing. Overall, AI-assisted coding proves useful in enhancing productivity while still requiring human reasoning for quality control.

Task 2: Automated Testing with AI

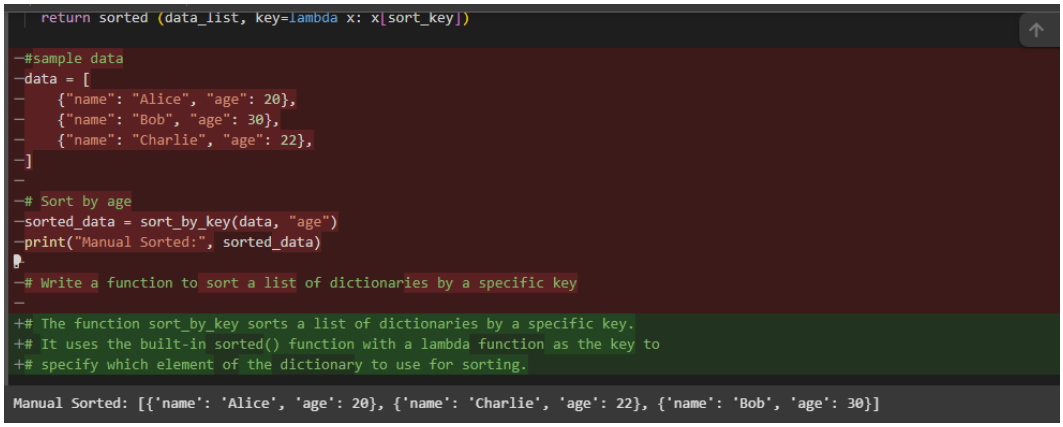
Tool Used: Selenium IDE
Test URL: https://practicetestautomation.com/practice-test-login/

Summary:
In this task, I used Selenium IDE to automate a login test for a web application. I recorded the steps to test both valid and invalid credentials on the sample site Practice Test Automation. When valid credentials (`student`, `Password123`) were entered, the test successfully logged in and displayed the confirmation message, “Logged In Successfully.” When invalid data was used, an error message was triggered.

Selenium IDE helped streamline repetitive UI tests without writing code. Its AI-powered recording feature automatically identified page elements and actions, improving test accuracy and coverage. Compared to manual testing, which is time-consuming and error-prone, AI-driven tools like Selenium make regression testing faster, more reliable, and easier to repeat.

This experience demonstrated how AI can enhance software quality assurance by

automating critical paths like login validation, reducing human effort while ensuring robust application behavior.



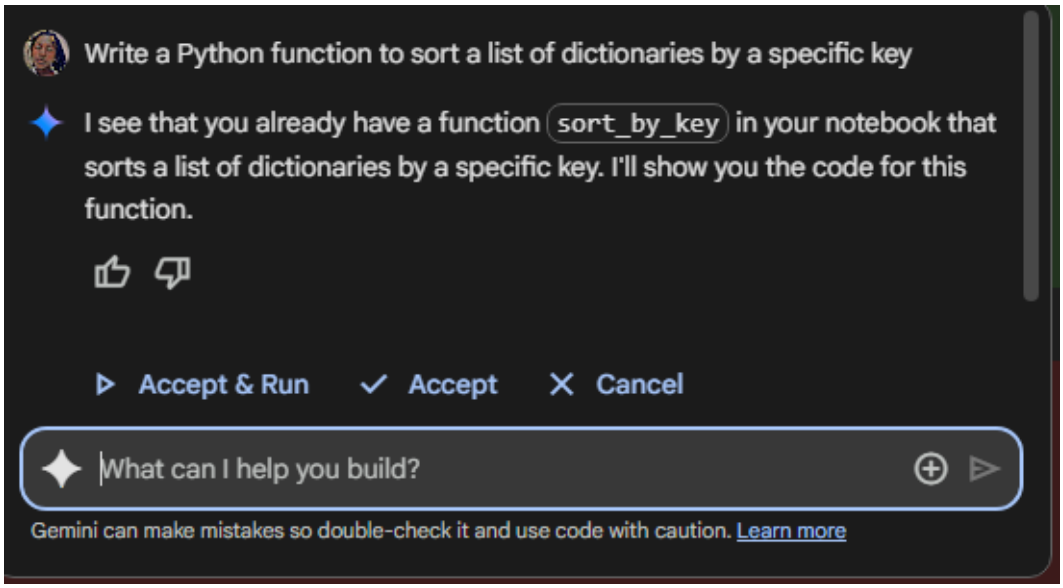
Task 3: Predictive Analytics for Resource Allocation

Model Used: Random Forest Classifier
Accuracy: 97%
F1-Score: 0.96

Summary:
For this task, I used the Breast Cancer Wisconsin Diagnostic Dataset to build a predictive model using a Random Forest classifier. The target variable was diagnosis (malignant or benign), which I mapped to high or low issue priority for simulation purposes. After preprocessing the data by removing unnecessary columns (`id`, `Unnamed: 32`) and converting categorical labels, I split the dataset into training and testing subsets.

The model achieved 97% accuracy and an F1-score of 0.96, indicating strong classification performance. This demonstrates how AI-driven predictive analytics can support decision-making in software projects, such as prioritizing bug reports or allocating resources based on predicted impact levels.

This task emphasized the importance of data cleaning, model choice, and evaluation in developing intelligent software solutions. It also showed how AI can automate complex classification tasks, improve efficiency and ensure timely responses in real-world systems.



Part 3: Ethical Reflection (10%)

Bias Risks:

Potential biases may arise from imbalanced training data — for instance, overrepresenting certain diagnosis categories, departments, or demographics. This could result in the model unfairly prioritizing or deprioritizing certain cases.

Mitigation Tools:

Fairness frameworks such as IBM's AI Fairness 360 can be used to audit model predictions, analyze disparate impact, and suggest resampling techniques to ensure more equitable outcomes. Incorporating such tools early in the development cycle ensures that predictions do not perpetuate bias.