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?

Faster code generation: AI-driven tools reduce the required time by developers on repetitive tasks, helping them focus on more complex and creative aspects of coding.

Error reduction: Users suggesting correct syntax and common patterns helps in error reduction, leading to fewer debugging cycles.

Reduced repetitive tasks: AI tools free developers from executing repetitive tasks. Instead, it lets them focus on more complex and creative tasks.

Limitations

Security concerns: The code generated by AI-driven code generation tools may not follow the best security practices, rendering the tool vulnerable. Thus, developers should be vigilant to ensure the used tool adheres to security standards.

Debugging and problem-solving: Developers still need to troubleshoot and debug their code, since the AI tool may not pinpoint the source of the error or provide the best solution.

Context sensitivity: The AI tool may fail to understand the project context, leading to suggestions that do not align with the overall project design or requirements.

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

Supervised	**Unsupervised**
Provides more accurate anomaly detection when there is sufficient labeled data	More flexible when the labeled data is limited or unavailable
It is limited to what is available in the training data	It may detect new types of anomalies not seen before
Classifies bugs, making interpretation easier	Hard to interpret the results as the model does not explain the code pattern

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

Reputational damage: Public knowledge of AI bias can lead to significant damage to an organization's brand and public perception.

Operational inefficiencies: Lead to poor decision-making, such as isolating specific market segments or excluding qualified candidates, negatively affecting the business outcomes.

Increase transparency: Helps build more transparent and accountable AI systems, which is fundamental for fostering user trust.

Perpetuate inequalities: AI systems can inherit and even amplify biases from their training data, leading to unfair treatment of marginalized groups.

2. Case Study Analysis

Read the article: *AI in DevOps: Automating Deployment Pipelines*.

How does AIOps improve software deployment efficiency?

It can predict potential failure before it happens. For instance, when a deployment for a change is done on the system, there is a high chance that during integration, conflicts may occur. Thus, AIOps has the capability of predicting a potential failure and rectifying it before it occurs, making the deployment process efficient.

AIOps ensures automated monitoring of the system. For instance, during deployment, the triggers can be set such that whenever there are changes in the system, it updates automatically. This minimizes human intervention, making the deployment process efficient.

Part 2: Practical Implementation (60%)

Task 1: AI-Powered Code Completion

- **Tool**: Use a code completion tool like GitHub Copilot or Tabnine.
- **Task**:
  - Write a Python function to sort a list of dictionaries by a specific key.
  - Compare the AI-suggested code with your manual implementation.
  - Document which version is more efficient and why.

Manual function

```python
from operator import itemgetter


def sort_by_category(dicts, key, reverse=False):
    """
    Sort the DataFrame by the 'Category' column.
    """

    return sorted(dicts, key=itemgetter(key), reverse=reverse)

# Sort the DataFrame by 'Category'
sorted_product = sort_by_category(products, "Product_Name", reverse=True)
for item in sorted_product:
    print(item)
```

✓ 0.0s Python

AI-Generated Code

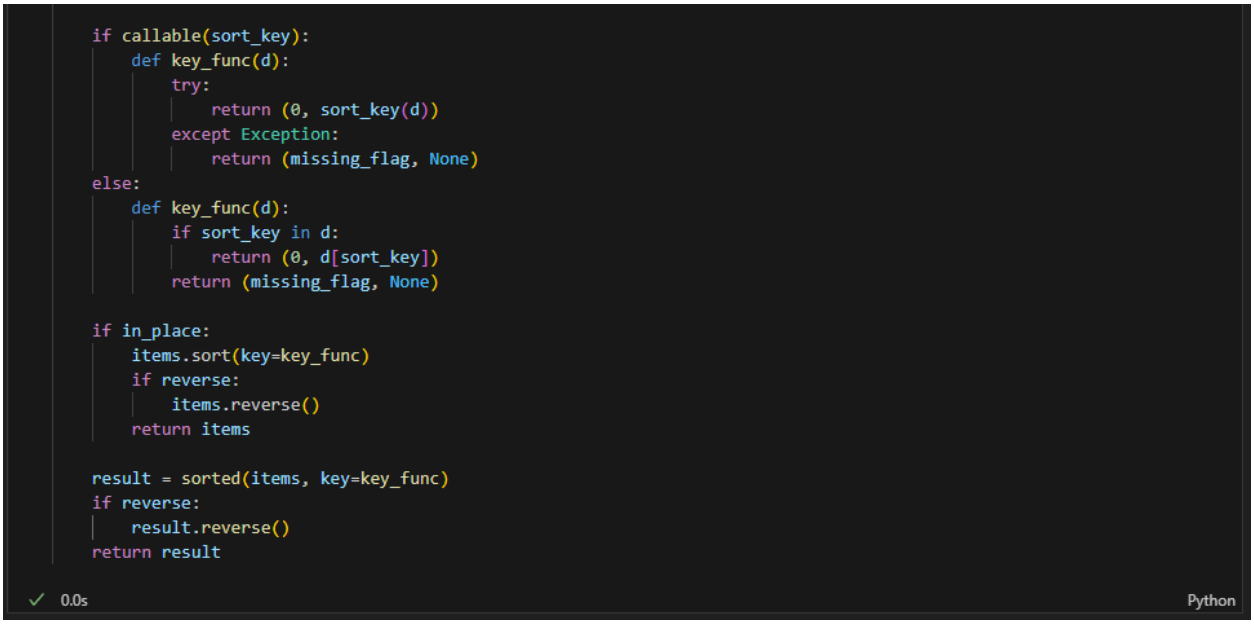
```python
def sort_list_of_dicts(items, sort_key, reverse=False, missing='last', in_place=False):
    """
    Sort a list of dictionaries by a specified key.

    Args:
        items (list): list of dicts to sort.
        sort_key (str | callable): dictionary key name or a callable that accepts a dict and returns the sort value.
        reverse (bool): whether to reverse the final order.
        missing (str): how to place items missing the key; 'last' (default) or 'first'.
        in_place (bool): if True, sort the provided list in place and return it; otherwise return a new list.

    Returns:
        list: sorted list of dictionaries (same object if in_place=True).

    Notes:
        - Items that lack the key are grouped at the start ('first') or end ('last') regardless of their values.
        - The function wraps extracted values in a tuple so missing-position behavior is stable and predictable.
    """
    if missing not in ('last', 'first'):
        raise ValueError("missing must be 'last' or 'first'")

    missing_flag = 1 if missing == 'last' else -1
```



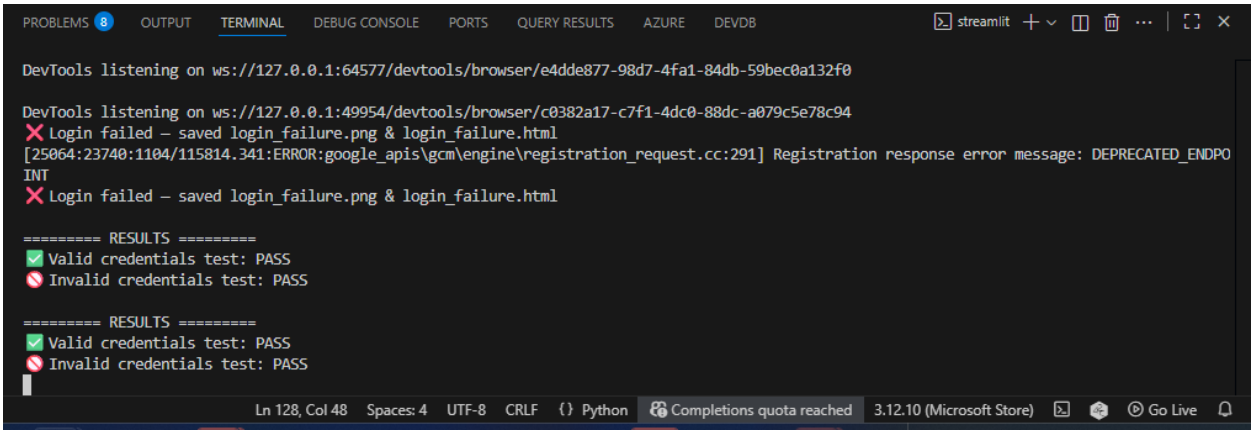
The AI-generated code is more efficient than the manual code because the AI-generated code takes care of any data fed into it. The manual implementation, on the other hand, is limited to a specific context. This makes the AI-generated code more efficient compared to the manually written code.

Task 2: Automated Testing with AI

- **Framework**: Use Selenium IDE with AI plugins or Testim.io.

Task 3: Predictive Analytics for Resource Allocation

- **Framework**: Use Selenium IDE with AI plugins or Testim.io.



AI ensures each test is conducted rigorously and efficiently, covering a wide range of tests, having identified patterns in code execution. Compared to manual testing, AI testing ensures continuous testing and adaptation to code changes.

Part 3: Ethical Reflection (10%)

- Potential biases in the dataset (e.g., underrepresented teams).

The model may become biased due to class imbalance. For instance, the benign cases numbered 357 compared to the malignant cases numbered 212. The model may become biased to the majority class, i.e., benign, leading to poor performance on the minority class. This means that the model may fail to recognize patterns in the minority class, leading to underfitting or high variance.

- How fairness tools like IBM AI Fairness 360 could address these biases.

Pre-processing method, such as Reweighing, can be utilized to assign higher weights to minority class instances to balance group representation. Additionally, adversarial networks can be used to minimize bias in predictions while preserving accuracy.