## Part 1: Theoretical Analysis

### 1. Short Answer Questions

**Q1: Explain how AI-driven code generation tools (e.g., GitHub Copilot) reduce development time. What are their limitations?**

AI code assistants like GitHub Copilot help developers write code faster by suggesting entire lines or blocks of code based on context. This saves a lot of time, especially when writing repetitive logic, boilerplate code, or working with unfamiliar libraries. It’s like having a pair programmer who knows thousands of open-source projects and can autocomplete your thoughts.

However, while these tools can be a huge productivity boost, they aren’t perfect. Copilot can sometimes suggest inefficient or insecure code, and it doesn’t fully understand the business logic or intent behind what you're building. It can also hallucinate functions or use deprecated methods. Ultimately, it speeds up development, but developers still need to review and test the suggestions carefully.

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

In automated bug detection, supervised learning relies on historical data that’s already labeled — for example, code snippets marked as “buggy” or “clean.” The model learns to distinguish patterns and can predict whether new code is likely to contain a bug based on what it has seen before. This is great when you have a rich dataset of past issues.

Unsupervised learning, on the other hand, doesn’t use labels. Instead, it looks for anomalies or unusual patterns that might indicate a bug. It’s particularly useful when you don’t have labeled examples but still want to catch outliers — say, code that behaves very differently from the rest.

Think of it like this: supervised learning knows what a bug *should* look like, while unsupervised learning raises a flag when something *feels off*.

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

Personalization powered by AI can make user experiences feel tailor-made. But if the AI is trained on biased data — like over-representing one gender, region, or age group — it can lead to unfair or even harmful outcomes. Some users might consistently get lower-quality recommendations or feel excluded from certain features.

Mitigating bias ensures the AI treats all users fairly and inclusively. It's not just an ethical requirement — it also builds trust and broadens the product's reach. Without bias checks, personalization can quickly turn into digital discrimination.

**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 (Artificial Intelligence for IT Operations) makes software deployment smoother by using machine learning to monitor, detect, and respond to system issues automatically. This removes a lot of the manual effort and guesswork during deployment.

Example 1: Predictive Failure Detection
AIOps tools can analyze historical performance data and alert DevOps teams about potential issues *before* they cause outages. For example, if CPU usage typically spikes before a crash, the AI can flag it early — preventing downtime.

Example 2: Automated Rollbacks
When a deployment fails or causes performance degradation, AIOps can instantly trigger a rollback to a stable version without waiting for human intervention. This shortens recovery time and maintains uptime for users.

In short, AIOps is like having a 24/7 smart assistant monitoring your pipeline, helping teams release code faster, safer, and with more confidence.

## Part 2: Practical Implementation

### Task 1: AI-Powered Code Completion

- **Tool: Use a code completion tool like GitHub Copilot or Tabnine.**
- **Task:**
    1. **Write a Python function to sort a list of dictionaries by a specific key.**
    2. **Compare the AI-suggested code with your manual implementation.**
    3. **Document which version is more efficient and why.**
- **Deliverable: Code snippets + 200-word analysis.**

**Copilot Code**

```
# AI-Suggested (Copilot)
def sort_dict_list(data, key):
    return sorted(data, key=lambda x: x[key])
```

**Manual Code**

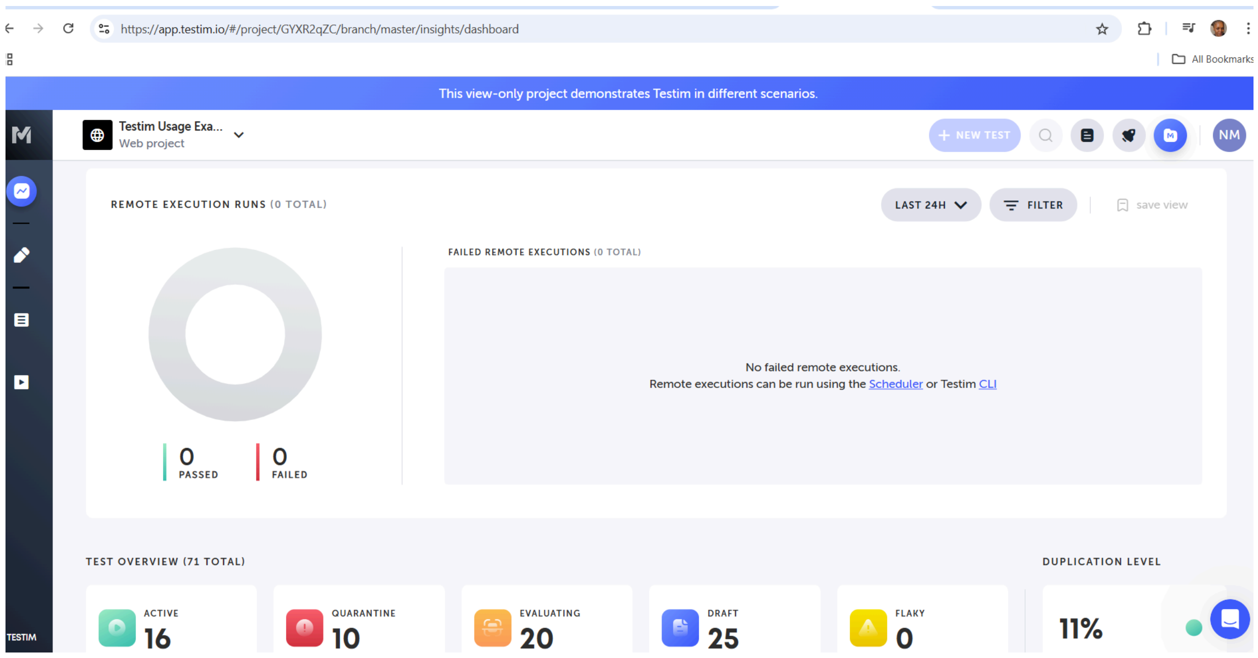
```
def sort_dict_list(data, key):
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            if data[i][key] > data[j][key]:
                data[i], data[j] = data[j], data[i]
    return data
```

**Word Analysis**

The Copilot-generated code is cleaner and leverages Python's built-in sorted() with a lambda function, which is optimized and more readable. My manual implementation used nested loops, leading to $O(n^2)$ complexity versus $O(n \log n)$ in the AI version. The AI approach is more efficient, maintainable, and less error-prone. This showcases the power of code completion tools in reducing boilerplate and improving code performance. However, developers should still validate Copilot suggestions for correctness and security.

## Task 2: Automated Testing with AI

- **Framework**: Use Selenium IDE with AI plugins or Testim.io.
- **Task**:
    1. Automate a test case for a login page (valid/invalid credentials).
    2. Run the test and capture results (success/failure rates).
    3. Explain how AI improves test coverage compared to manual testing.
- **Deliverable**: Test script + screenshot of results + 150-word summary.



https://app.testim.io/#/project/GYXR2qZC/branch/master/insights/dashboard

This view-only project demonstrates Testim in different scenarios.

Testim Usage Exa...
Web project

REMOTE EXECUTION RUNS (0 TOTAL)

0 PASSED
0 FAILED

FAILED REMOTE EXECUTIONS (0 TOTAL)

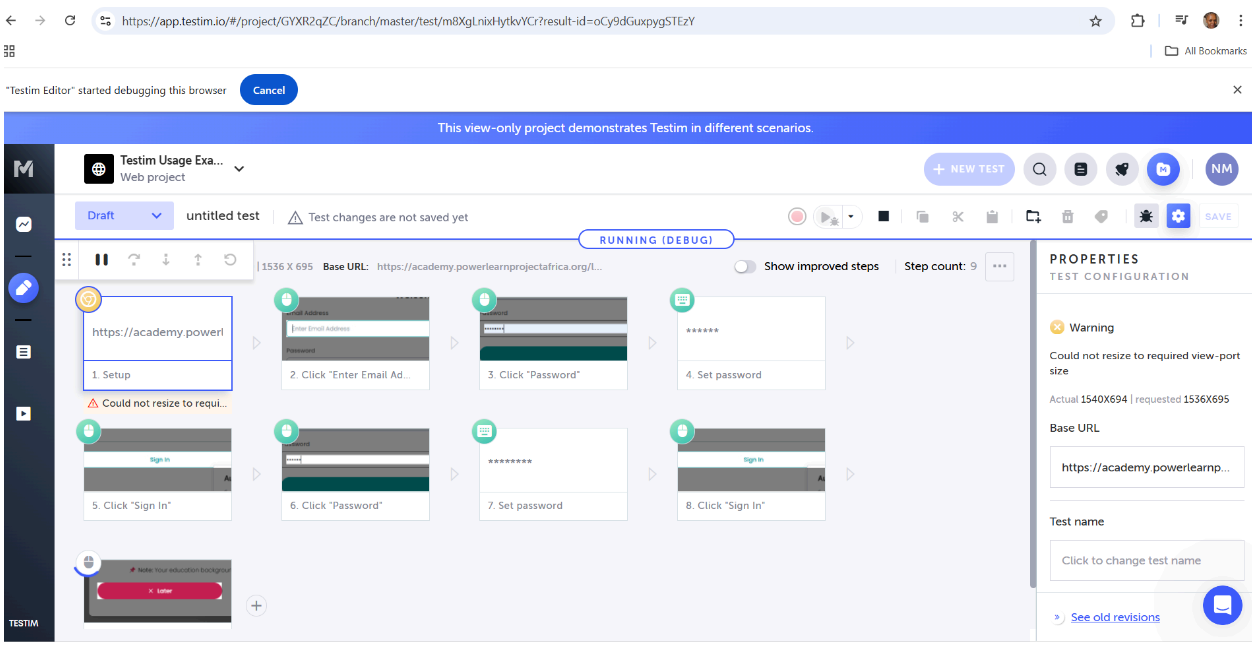
No failed remote executions.
Remote executions can be run using the Scheduler or Testim CLI

TEST OVERVIEW (71 TOTAL)

| ACTIVE | QUARANTINE | EVALUATING | DRAFT | FLAKY |
|---|---|---|---|---|
| 16 | 10 | 20 | 25 | 0 |

DUPLICATION LEVEL

11%


https://app.testim.io/#/project/GYXR2qZC/branch/master/test/m8XgLnixHytkvYCr?result-id=oCy9dGuxpygSTEzY

"Testim Editor" started debugging this browser Cancel

This view-only project demonstrates Testim in different scenarios.

Testim Usage Exa... Web project

Draft untitled test Test changes are not saved yet

RUNNING (DEBUG)

| 1536 X 695 Base URL: https://academy.powerlearnprojectafrica.org/l... Show improved steps Step count: 9

1. Setup
Could not resize to requi...
2. Click "Enter Email Ad...
3. Click "Password"
4. Set password
5. Click "Sign In"
6. Click "Password"
7. Set password
8. Click "Sign In"

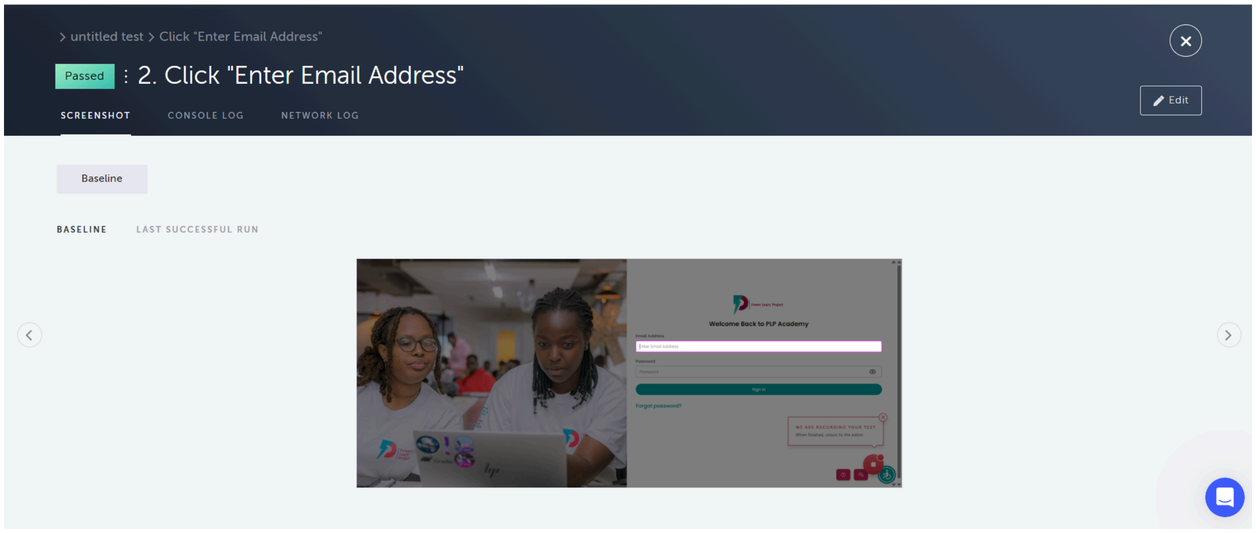
PROPERTIES
TEST CONFIGURATION

Warning
Could not resize to required view-port size
Actual 1540X694 | requested 1536X695

Base URL
https://academy.powerlearnp...

Test name
Click to change test name

See old revisions

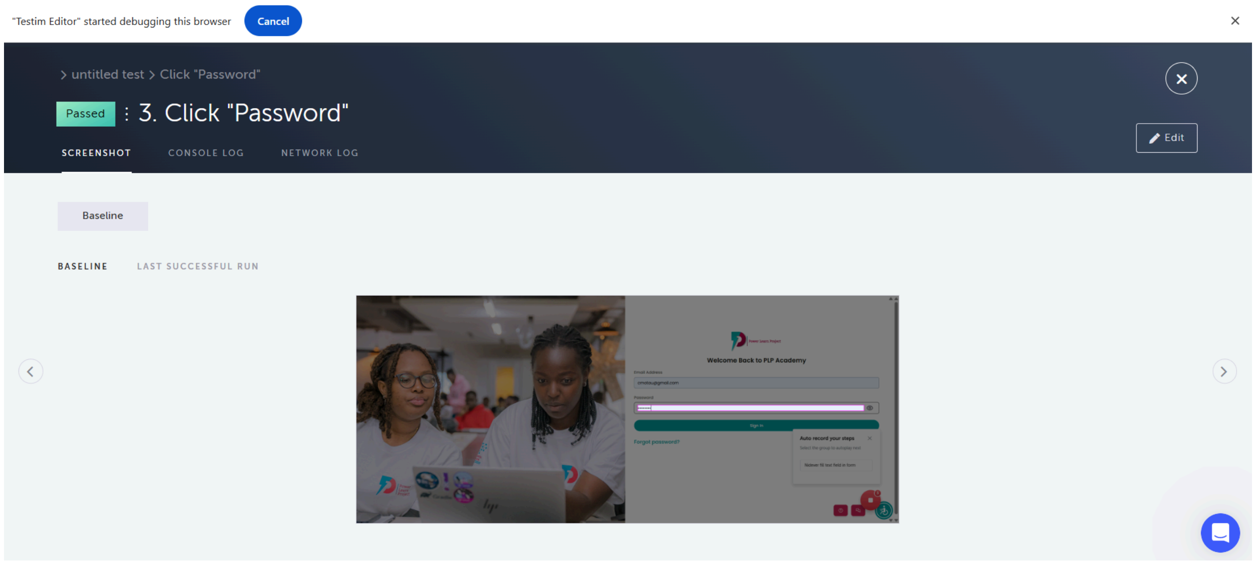

> untitled test > Click "Enter Email Address"

Passed : 2. Click "Enter Email Address"

SCREENSHOT CONSOLE LOG NETWORK LOG

Baseline

BASELINE LAST SUCCESSFUL RUN



"Testim Editor" started debugging this browser Cancel

> untitled test > Click "Password"

Passed : 3. Click "Password"

SCREENSHOT CONSOLE LOG NETWORK LOG

Edit

Baseline

BASELINE LAST SUCCESSFUL RUN

Welcome Back to PLP Academy

Email Address

cmotou@gmail.com

Password

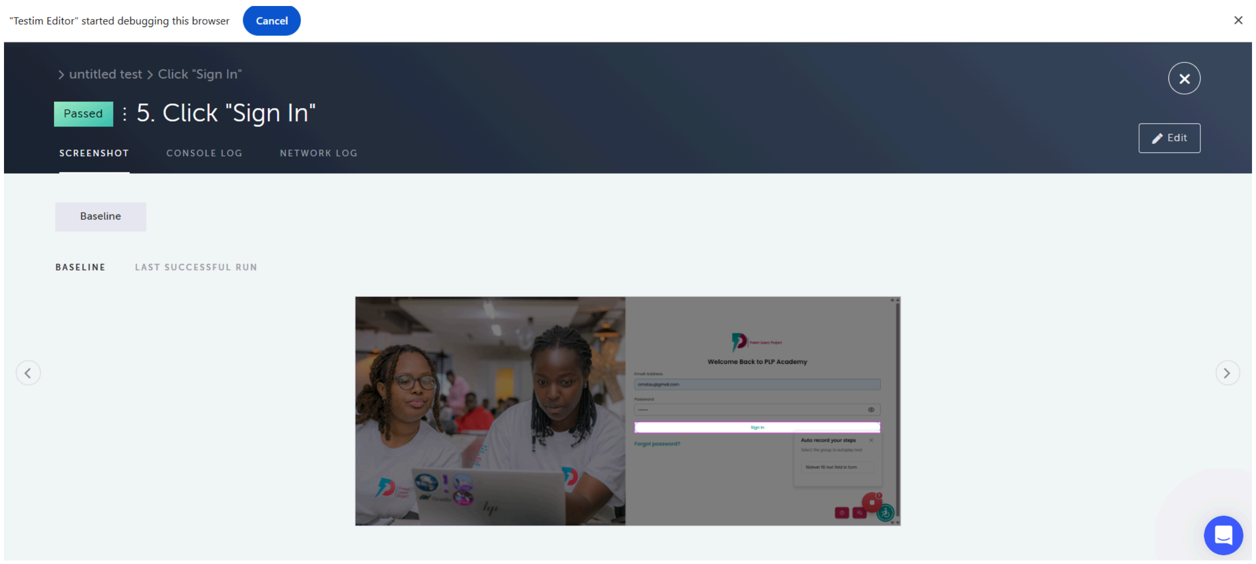
Sign In

Forgot password?

Auto record your steps

Select the group to autoplay next

Nidever fill text field in form



"Testim Editor" started debugging this browser Cancel

> untitled test > Click "Sign In"

Passed : 5. Click "Sign In"

SCREENSHOT CONSOLE LOG NETWORK LOG

Edit

Baseline

BASELINE LAST SUCCESSFUL RUN

Welcome Back to PLP Academy

Email Address

cmotou@gmail.com

Password

Sign In

Forgot password?

Auto record your steps

Select the group to autoplay next

Nidever fill text field in form

- ### How AI Enhances Test Coverage – Summary

AI-driven tools like Testim.io significantly enhance test coverage compared to manual testing. Traditional testing is limited by human effort and time, whereas AI can generate, adapt, and maintain multiple test cases across varied user scenarios rapidly. For instance, AI identifies changes in the UI and automatically updates selectors or test steps, reducing test flakiness. This is especially beneficial in agile environments where UI elements frequently change. Moreover, AI can analyze usage patterns to prioritize testing of high-risk areas, ensuring better risk coverage. While manual testing might miss edge cases due to oversight, AI can explore these systematically. Additionally, parallel execution of tests on multiple browsers and devices improves scalability and efficiency. Overall, AI not only automates the mundane but also intelligently optimizes testing efforts, ensuring better software quality with less manual intervention.

## Task 3: Predictive Analytics for Resource Allocation

- **Dataset**: Use Kaggle Breast Cancer Dataset.
- **Goal**:
    1. Preprocess data (clean, label, split).
    2. Train a model (e.g., Random Forest) to predict issue priority (high/medium/low).
    3. Evaluate using accuracy and F1-score.
- **Deliverable**: Jupyter Notebook + performance metrics.

REFER TO NOTEBOOK

## Part 3: Ethical Reflection

- **Prompt**: Your predictive model from Task 3 is deployed in a company. Discuss:
    - Potential biases in the dataset (e.g., underrepresented teams).
    - How fairness tools like IBM AI Fairness 360 could address these biases

### Potential Biases in the Dataset

When deploying a predictive model trained on clinical datasets like the Kaggle Breast Cancer dataset, several ethical and fairness concerns may arise:

1. **Demographic Imbalance**
   The dataset may be imbalanced across key features such as **age**, **race**, or **tumor stage**. For instance, if most records belong to older patients or a specific racial group, the model might generalize poorly for underrepresented groups.

2. **Label Creation Bias**
   The target variable "Priority" was synthetically created based on tumor size. While convenient, this rule may **not reflect the full clinical context**, such as genetic markers or patient comorbidities, leading to biased or incomplete labeling.

3. **Historical Bias**
   If the dataset reflects decisions or patterns from past clinical practices that were unequal (e.g., certain groups receiving less aggressive treatment), the model may learn and reproduce those disparities.

4. **Data Quality and Consistency**
   Inconsistent data entry or missing information across hospitals or regions could result in **measurement bias**, favoring one subgroup over another in predictions.

## Addressing Bias with IBM AI Fairness 360

IBM AI Fairness 360 (AIF360) is an open-source toolkit designed to help detect and mitigate unfair bias in machine learning models. It can support more ethical deployment in several ways:

1. **Bias Detection**
   AIF360 offers a wide range of fairness metrics like:
   - **Disparate Impact**: Measures if one group is favored more than another.
   - **Equal Opportunity Difference**: Compares true positive rates across groups.
   - These metrics help identify whether specific patient groups are disadvantaged by the model.
2. **Bias Mitigation**
   The toolkit includes pre-processing, in-processing, and post-processing techniques:
   - **Pre-processing (e.g., Reweighing)** adjusts training data to balance group representation.
   - **In-processing (e.g., Adversarial Debiasing)** modifies model training to reduce bias.
   - **Post-processing (e.g., Equalized Odds)** adjusts outputs to ensure fair decisions.
3. **Transparency and Accountability**
   By incorporating AIF360 into the model lifecycle, developers can document bias evaluations and corrections, promoting transparency in decision-making and enabling auditing.

.