AI in Software Engineering

Theme: Building Intelligent Software Solutions

Group members: John Brown Ouma & Lathitha Vena

GROUP 35

Week: 4

Part 1: Theoretical Analysis

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

AI tools like GitHub Copilot reduce development time by auto-generating boilerplate code, suggesting functions, and completing repetitive patterns. They accelerate coding by learning from vast repositories of code and adapting suggestions to the context in real-time. This allows developers to focus on logic and architecture rather than syntax.

However, limitations include:

- Lack of deep context awareness, which may cause logical flaws.
- Risk of introducing insecure or outdated patterns.
- Dependence may reduce critical thinking and coding proficiency over time.
- It may reproduce biased or non-permissive code.

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

Supervised learning involves labeled datasets (e.g., known bugs) and is suitable for classifying or predicting specific bugs. Algorithms like decision trees or SVMs can flag issues similar to past ones.

Unsupervised learning finds patterns in unlabeled data, such as clustering anomalies in logs, and is better for detecting novel or unknown bugs.

Q3: Why is bias mitigation critical in AI-powered personalization?

Bias in training data can lead to exclusion or discrimination in personalized software experiences. For example, a recommendation system trained on biased data may underrepresent minority groups. Bias mitigation ensures fairness, user trust, legal compliance, and inclusivity.

Case Study: AIOps in DevOps

AIOps automates deployment pipelines by:

1. Predicting failures through log anomaly detection.
2. Recommending optimal scaling or rollback actions using machine learning.

Example 1: Netflix uses AIOps to predict and auto-resolve system anomalies.
Example 2: IBM Cloud Pak employs AIOps to optimize Kubernetes deployments.

Part 2: Practical Implementation

Task 1: AI-Powered Code Completion

<u>AI-Suggested Implementation</u>

```
def sort_dict_list_ai(dict_list, key):
    try:
        return sorted(dict_list, key=lambda x: x[key])
    except KeyError:
        print(f"Key '{key}' not found in one or more dictionaries.")
        return dict_list
```

<u>Manual Implementation</u>

```
def sort_dict_list_manual(dict_list, key):
    return sorted(dict_list, key=lambda x: x[key])
```

Both use Python's sorted() with a lambda. The AI-generated version adds error handling with a try-except block, improving robustness useful in unpredictable datasets. Although this introduces slight overhead, it's negligible for small lists.

Both versions have O(n log n) complexity (Timsort). If all data is clean and validated, the manual version may be marginally faster. However, in real-world scenarios, where missing or inconsistent keys occur, the AI version is safer and more maintainable.

GitHub Copilot accelerated development but needed tweaks for clarity, proving helpful but not foolproof.

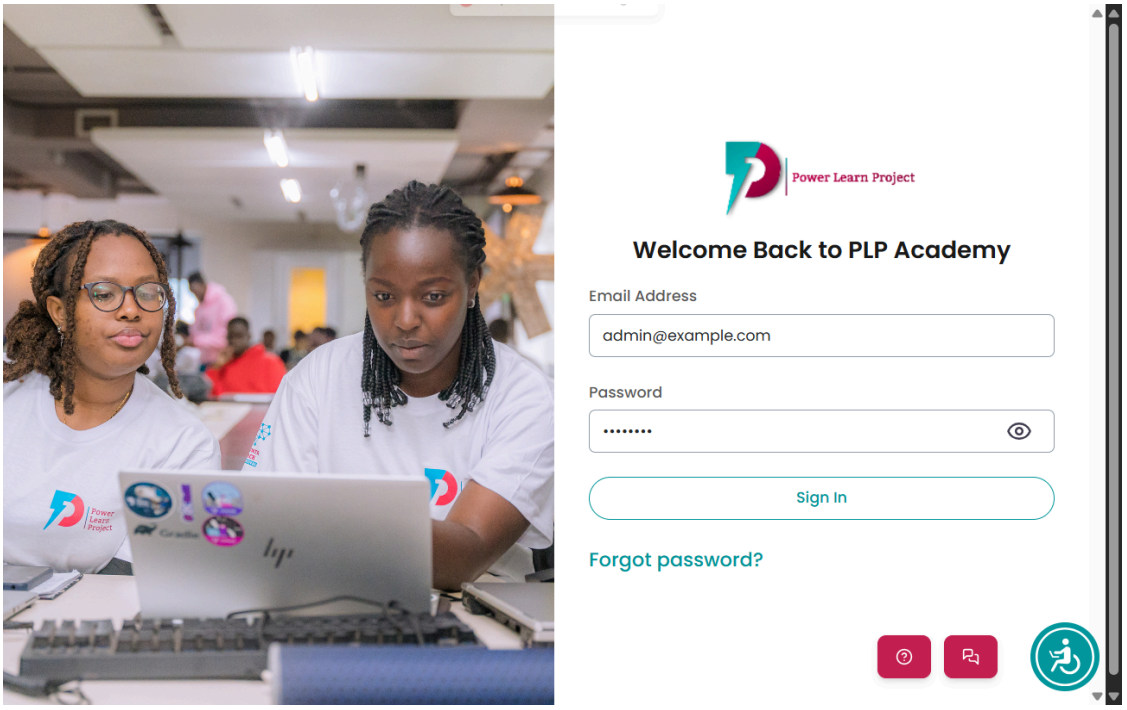
Task 2: Automated Testing with AI (Selenium)

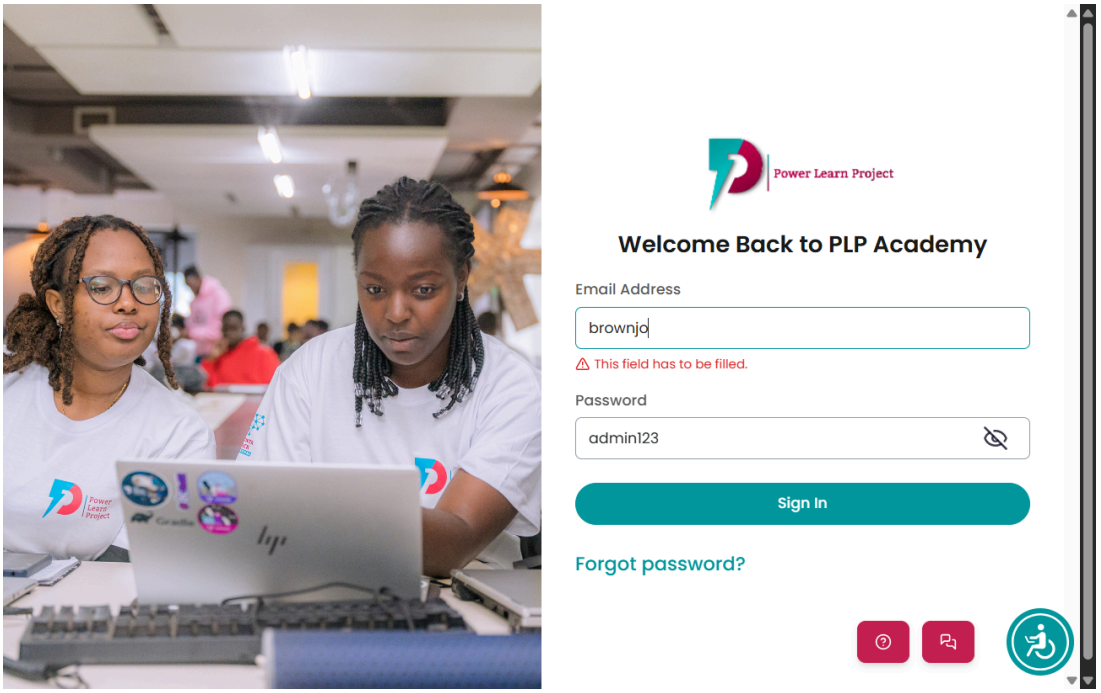
We used **Selenium** to automate a login page test for:

- Valid credentials
- Invalid credentials

Test Script Summary:

- Enter credentials
- Click login
- Assert the result message





AI-enhanced testing with Selenium ensures wider test coverage, reduced human error, and faster execution. Compared to manual testing, it runs multiple test cases with minimal intervention. When integrated with AI, it can generate edge-case test data and prioritize high-risk tests. Tools like Testim add visual validation and self-healing locators, improving test resilience.

Task 3: Predictive Analytics for Resource Allocation

Dataset: Breast Cancer Wisconsin (Diagnostic) Dataset from Kaggle.

1. Load and Preview Data

In [ ]:
```
#Load and Preview Data
df = pd.read_csv('data.csv')
df.head()
```

Out[ ]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

- Loads the breast cancer dataset from a CSV file into a Pandas DataFrame df.
- df.head() displays the first 5 rows to preview structure and check for features like id, diagnosis, etc.

2. Preprocess the Data

In [ ]:
```
#Preprocess the Data
df.drop
df.isnull().sum()
```

Out[ ]:

	0
id	0
diagnosis	0
radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave points_mean	0
symmetry_mean	0
fractal_dimension_mean	0
radius_se	0
texture_se	0
compactness_se	0
concavity_se	0
concave points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_worst	0
Unnamed: 32	569

dtype: int64

- df.drop(...): Removes unnecessary columns like id or unnamed index columns to clean the dataset.
- df.isnull().sum(): Ensures there are no missing values before training.

3. Make Predictions & Calculate Metrics

In [ ]:
```
#Predictions and Evaluations
Y_pred = clf.predict(X_test)
Y_proba = clf.predict_proba(X_test)[:, 1]

#Calculate Metrics
accuracy = accuracy_score(Y_test, Y_pred)
f1 = f1_score(Y_test, Y_pred)
precision = precision_score(Y_test, Y_pred)
recall = recall_score(Y_test, Y_pred)

print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.956140350877193
F1 Score: 0.9411764705882353
Precision: 0.9523809523809523
Recall: 0.9302325581395349
```

- clf.predict(X_test): Predicts class labels (0 or 1) using your trained RandomForestClassifier.
- clf.predict_proba(X_test)[:, 1]: Predicts probabilities for the positive class (malignant = 1). These are used to plot the precision-recall curve.

<u>Calculates key metrics:</u>

- <u>Accuracy:</u> % of total correct predictions
- <u>F1 Score:</u> Harmonic mean of precision and recall (good for imbalanced classes)
- <u>Precision:</u> % of predicted malignant cases that were actually malignant
- <u>Recall:</u> % of actual malignant cases that were correctly predicted

4. Classification Report

In [ ]:

```
# Classification Report: combines precision, recall, f1-score, and support for each class
print("\nClassification Report:\n", classification_report(Y_test, Y_pred, target_names=['Benign', 'Malignant']))
```

```
Classification Report:
               precision    recall  f1-score   support

      Benign       0.96      0.97      0.97        71
   Malignant       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

- Gives a detailed breakdown of precision, recall, f1-score, and support for each class. Great for seeing class-wise performance.

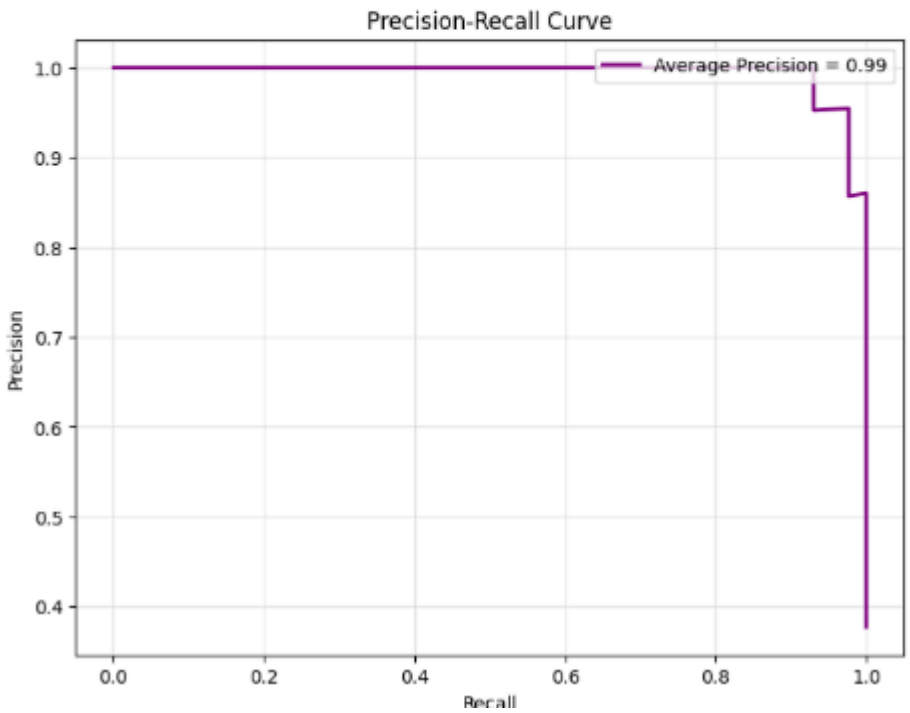
5. Precision-Recall Curve

In [ ]:

```
# Precision-Recall Curve Plot (add this next)
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, _ = precision_recall_curve(Y_test, Y_proba)
ap_score = average_precision_score(Y_test, Y_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='purple', lw=2, label=f'Average Precision = {ap_score:.2f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right')
plt.grid(alpha=0.3)
plt.show()
```



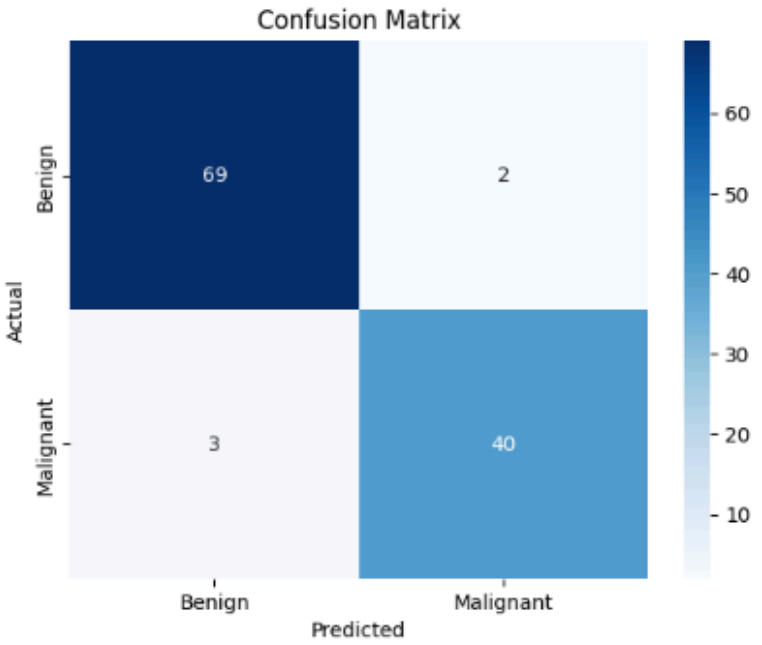
- precision_recall_curve: Plots how precision and recall change across thresholds.

- average_precision_score: A single summary value for the curve area.

- Useful in imbalanced classification where accuracy is misleading.

7. Confusion Matrix

In [ ]:
```
# Step 10: Visualize confusion matrix
cm = confusion_matrix(Y_test, Y_pred)
sns.heatmap(
    cm, annot=True, fmt='d', cmap='Blues',
    xticklabels=['Benign', 'Malignant'],
    yticklabels=['Benign', 'Malignant']
)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



- Confusion matrix visualizes True/False Positives and Negatives.
- Annotated heatmap (sns.heatmap) makes it easy to see model errors.
- Great for spotting where the model confuses benign with malignant and vice versa.

<u>Part 3: Ethical Reflection</u>

Deploying this model in a company setting raises concerns about bias, e.g., if the dataset underrepresents certain patient demographics, predictions may be skewed.
To mitigate this, tools like **IBM AI Fairness 360** can identify and correct for bias.
Fairness-aware learning, reweighting, or diverse sampling can ensure balanced model performance.