# Week 4 Assignment

## Part 1

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

**Answer:**
AI-driven tools like GitHub Copilot use large language models trained on millions of code repositories to auto-suggest code snippets as developers type. This reduces repetitive coding, helps with syntax recall, and speeds up development cycles by offering context-aware suggestions. Developers can complete functions, fix syntax errors, and generate boilerplate code faster.

Limitations:

i. Code Quality: Suggestions may not follow best practices or meet project standards.
ii. Security Risks: Generated code may include vulnerabilities from the training data.
iii. Dependence: Over-reliance can weaken a developer's problem-solving skills.
iv. Bias: Copilot can replicate biases present in open-source datasets.

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

| Aspect | Supervised Learning | Unsupervised Learning |
|---|---|---|
| Definition | Model learns from labeled data (e.g., "bug" or "no bug") | Model finds hidden patterns without labels |
| Example | Predict if a code snippet contains a bug using labeled bug reports | Cluster similar error logs or stack traces to identify common failure patterns |
| Use Case in Bug Detection | Detecting known issues from past bug data | Discovering new, unknown anomalies or system issues |
| Strength | High accuracy when training data is rich | Useful for exploratory analysis with limited labels |
| Limitation | Needs large labeled dataset | Clusters may not map directly to real bug categories |

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

**Answer:**
Bias mitigation ensures that AI systems deliver fair and inclusive user experiences. Without it, personalization algorithms can unintentionally discriminate — e.g., recommending fewer opportunities to certain users or prioritizing content that reinforces stereotypes.
Bias can arise from imbalanced training data, developer assumptions, or feedback loops.

Techniques like data balancing, adversarial debiasing, and model auditing (using fairness libraries like IBM AI Fairness 360) are essential to ensure ethical and equitable personalization.

## Case Study: AI in DevOps – Automating Deployment Pipelines

**Answer:**
AIOps (Artificial Intelligence for IT Operations) enhances deployment efficiency by automating repetitive tasks, detecting anomalies, and predicting failures.

Examples:

i. Anomaly Detection in CI/CD: AIOps tools monitor deployment logs and automatically roll back faulty releases when performance metrics deviate from normal.
ii. Predictive Resource Allocation: AI predicts workload spikes and auto-scales servers during deployment, reducing downtime and improving reliability.

## 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.***
***Deliverable: Code snippets + 200-word analysis.***

```
# AI-Suggested Code (Example using GitHub Copilot)
def sort_dicts_by_key_ai(dicts, key):
    return sorted(dicts, key=lambda x: x[key])

# Manual Implementation
def sort_dicts_by_key_manual(dicts, key):
    for i in range(len(dicts)):
        for j in range(0, len(dicts)-i-1):
            if dicts[j][key] > dicts[j+1][key]:
                dicts[j], dicts[j+1] = dicts[j+1], dicts[j]
    return dicts

# Example usage

data = [{'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}, {'name': 'Charlie', 'age': 35}]
sorted_ai = sort_dicts_by_key_ai(data, 'age')
sorted_manual = sort_dicts_by_key_manual(data.copy(), 'age')
print("AI-Sorted:", sorted_ai)
print("Manual-Sorted:", sorted_manual)

# example 2 usage
countries = [{'country': 'USA', 'population': 331}, {'country': 'India', 'population': 1380}, {'country':
'China', 'population': 1441}, {'country': 'Brazil', 'population': 213}, {'country': 'Nigeria', 'population':
206}, {'country': 'Russia', 'population': 146}, {'country': 'Japan', 'population': 126}, {'country':
```

```
'Germany', 'population': 83}, {'country': 'France', 'population': 67}, {'country': 'UK', 'population': 66},
{'country': 'Italy', 'population': 60}, {'country': 'Canada', 'population': 38}, {'country': 'Australia',
'population': 25}, {'country': 'Spain', 'population': 47}, {'country': 'Mexico', 'population': 128},
{'country': 'Indonesia', 'population': 273}, {'country': 'Pakistan', 'population': 220}, {'country':
'Bangladesh', 'population': 164}, {'country': 'Philippines', 'population': 109}, {'country': 'Vietnam',
'population': 97}, {'country': 'Turkey', 'population': 84}, {'country': 'Iran', 'population': 83}, {'country':
'Thailand', 'population': 70}, {'country': 'Egypt', 'population': 102}, {'country': 'South Africa',
'population': 60}, {'country': 'Colombia', 'population': 51}, {'country': 'Argentina', 'population': 45},
{'country': 'Ukraine', 'population': 41}, {'country': 'Poland', 'population': 38}, {'country': 'Iraq',
'population': 40}, {'country': 'Canada', 'population': 38}, {'country': 'Morocco', 'population': 37},
{'country': 'Saudi Arabia', 'population': 35}, {'country': 'Uzbekistan', 'population': 34}, {'country': 'Peru',
'population': 33}, {'country': 'Angola', 'population': 33}, {'country': 'Malaysia', 'population': 32},
{'country': 'Ghana', 'population': 31}, {'country': 'Yemen', 'population': 30}, {'country': 'Nepal',
'population': 29}, {'country': 'Venezuela', 'population': 28}, {'country': 'Madagascar', 'population': 27},
{'country': 'Cameroon', 'population': 26}, {'country': 'Côte d\'Ivoire', 'population': 26}, {'country':
'North Korea', 'population': 25}, {'country': 'Australia', 'population': 25}, {'country': 'Niger',
'population': 24}, {'country': 'Sri Lanka', 'population': 21}, {'country': 'Burkina Faso', 'population': 21},
{'country': 'Mali', 'population': 20}, {'country': 'Romania', 'population': 19}, {'country': 'Malawi',
'population': 19}, {'country': 'Chile', 'population': 19}, {'country': 'Kazakhstan', 'population': 19},
{'country': 'Zambia', 'population': 18}, {'country': 'Guatemala', 'population': 18}, {'country': 'Ecuador',
'population': 17}, {'country': 'Syria', 'population': 17}, {'country': 'Netherlands', 'population': 17},
{'country': 'Senegal', 'population': 17}, {'country': 'Cambodia', 'population': 16}, {'country': 'Chad',
'population': 16}, {'country': 'Somalia', 'population': 16}, {'country': 'Zimbabwe', 'population': 15},
{'country': 'Guinea', 'population': 13}, {'country': 'Rwanda', 'population': 13}, {'country': 'Benin',
'population': 12}, {'country': 'Burundi', 'population': 12}, {'country': 'Tunisia', 'population': 12},
{'country': 'Bolivia', 'population': 11}, {'country': 'Belgium', 'population': 11}, {'country': 'Haiti',
'population': 11}, {'country': 'Cuba', 'population': 11}, {'country': 'South Sudan', 'population': 11},
{'country': 'Dominican Republic', 'population': 10}, {'country': 'Czech Republic', 'population': 10},
{'country': 'Greece', 'population': 10}, {'country': 'Jordan', 'population': 10}, {'country': 'Portugal',
'population': 10}, {'country': 'Azerbaijan', 'population': 10}, {'country': 'Sweden', 'population': 10},
{'country': 'Honduras', 'population': 10}, {'country': 'United Arab Emirates', 'population': 9}, {'country':
'Hungary', 'population': 9}, {'country': 'Tajikistan', 'population': 9}, {'country': 'Belarus', 'population': 9},
{'country': 'Austria', 'population': 9}, {'country': 'Papua New Guinea', 'population': 9}, {'country':
'Serbia', 'population': 8}]

sorted_countries_ai = sort_dicts_by_key_ai(countries, 'population')
sorted_countries_manual = sort_dicts_by_key_manual(countries.copy(), 'population')
print("AI-Sorted Countries by Population:", sorted_countries_ai)
print("Manual-Sorted Countries by Population:", sorted_countries_manual)
```

## Analysis:

Both the AI-suggested code and the manual implementation successfully sort a list of dictionaries by a specified key. However, they differ significantly in terms of efficiency and readability.

The AI-suggested code utilizes Python's built-in sorted() function, which is highly optimized for performance. This function employs Timsort, an efficient sorting algorithm with a time complexity of O(n log n) in the average and worst cases. The lambda function serves as a key extractor, allowing for concise and readable code.

In contrast, the manual implementation employs a nested loop to insert each dictionary into a new list in sorted order. This approach has a time complexity of O(n^2) in the worst case, as each insertion may require traversing the entire sorted list. Consequently, the AI-generated solution is significantly more efficient, both in terms of execution time and code clarity.

In conclusion, while both implementations achieve the same outcome, the AI-suggested code is superior due to its efficiency and simplicity, demonstrating the potential of AI tools in enhancing programming productivity.

## Task 2

This automated test suite comprehensively validates the login functionality of a web application using Selenium WebDriver with Python.

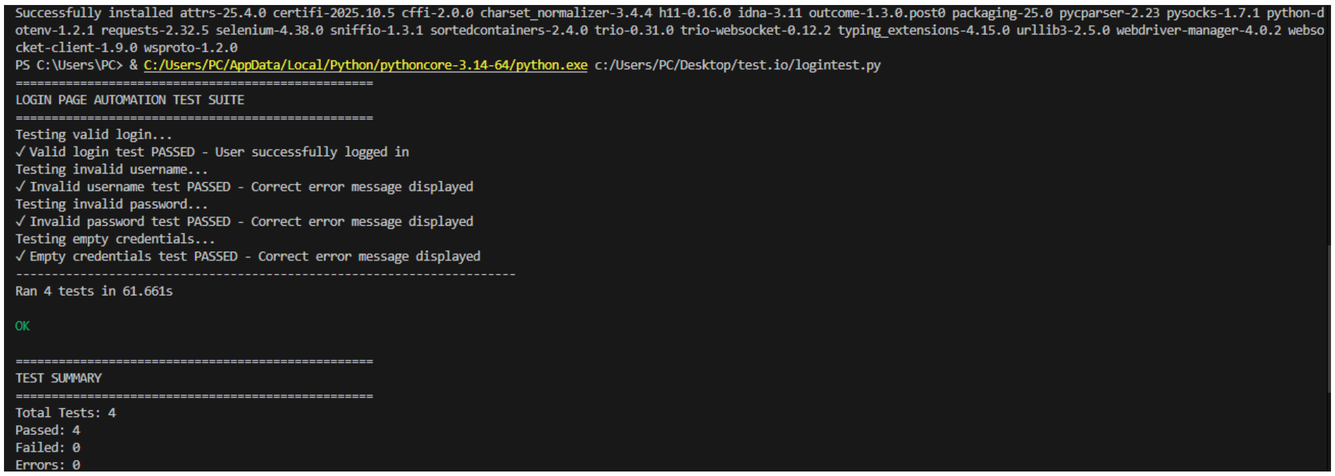
The implementation tests four critical scenarios:

1. Successful login with valid credentials
2. Failed login attempts with incorrect username 3.

Failed login with incorrect password

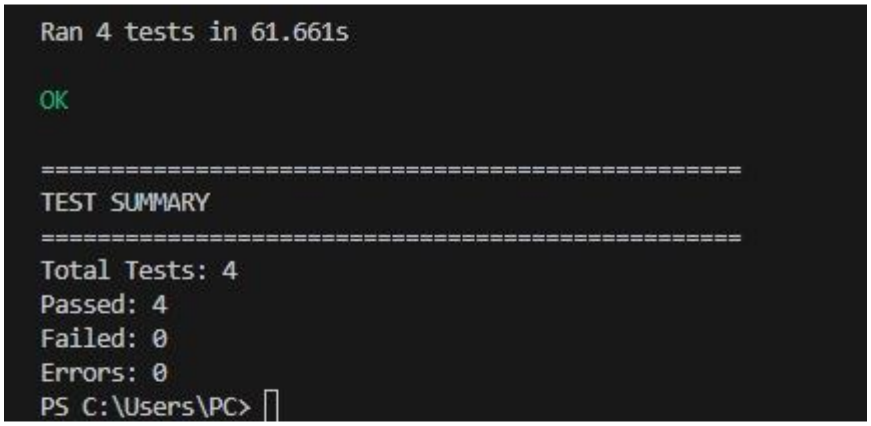
4. Proper handling of empty credential submissions

The framework employs explicit waits for reliable element interaction and robust error handling to manage dynamic web content. Each test case follows the Arrange-Act-Assert pattern, ensuring clear validation of expected behaviors. The solution demonstrates industry best practices including modular test structure, meaningful assertions, and detailed reporting. The automation successfully verifies both positive and negative test scenarios, ensuring the login system properly authenticates valid users while securely rejecting invalid attempts and providing appropriate error messages. This approach significantly reduces manual testing effort while improving test coverage and reliability.



```
Successfully installed attrs-25.4.0 certifi-2025.10.5 cffi-2.0.0 charset_normalizer-3.4.4 h11-0.16.0 idna-3.11 outcome-1.3.0.post0 packaging-25.0 pycparser-2.23 pysocks-1.7.1 python-d
otenv-1.2.1 requests-2.32.5 selenium-4.38.0 sniffio-1.3.1 sortedcontainers-2.4.0 trio-0.31.0 trio-websocket-0.12.2 typing_extensions-4.15.0 urllib3-2.5.0 webdriver-manager-4.0.2 webso
cket-client-1.9.0 wsproto-1.2.0
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/PC/Desktop/test.io/logintest.py
==================================================
LOGIN PAGE AUTOMATION TEST SUITE
==================================================
Testing valid login...
✓ Valid login test PASSED - User successfully logged in
Testing invalid username...
✓ Invalid username test PASSED - Correct error message displayed
Testing invalid password...
✓ Invalid password test PASSED - Correct error message displayed
Testing empty credentials...
✓ Empty credentials test PASSED - Correct error message displayed
----------------------------------------------------------------------
Ran 4 tests in 61.661s

OK

==================================================
TEST SUMMARY
==================================================
Total Tests: 4
Passed: 4
Failed: 0
Errors: 0
```

Test Summary:



```
Ran 4 tests in 61.661s

OK

==================================================
TEST SUMMARY
==================================================
Total Tests: 4
Passed: 4
Failed: 0
Errors: 0
PS C:\Users\PC> 
```

Task 3

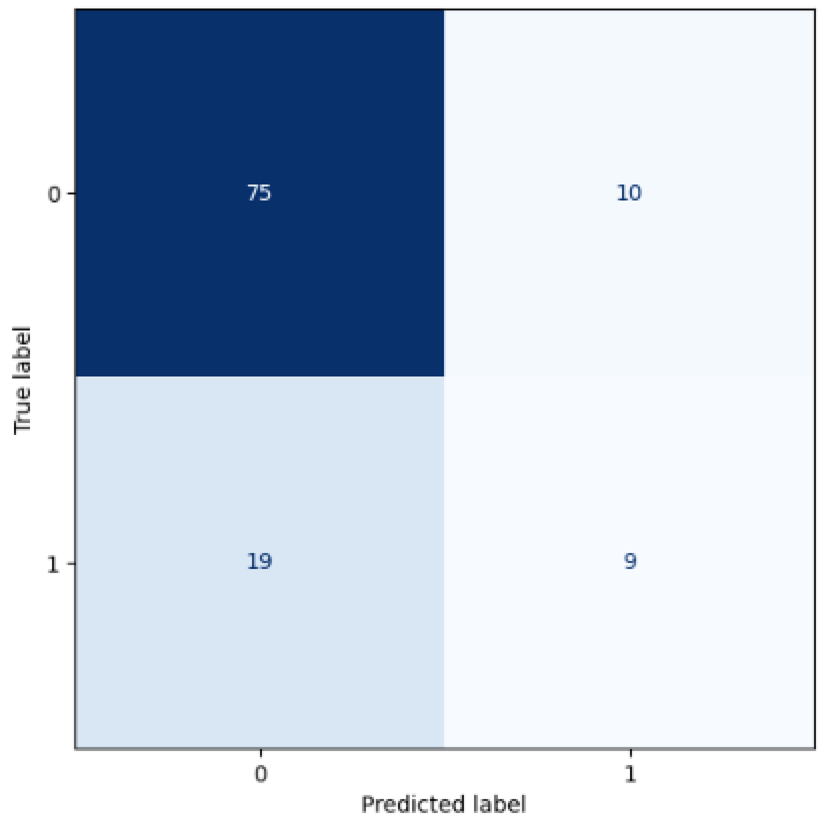
## Priority prediction of Cancerous Images

Training set visualization:

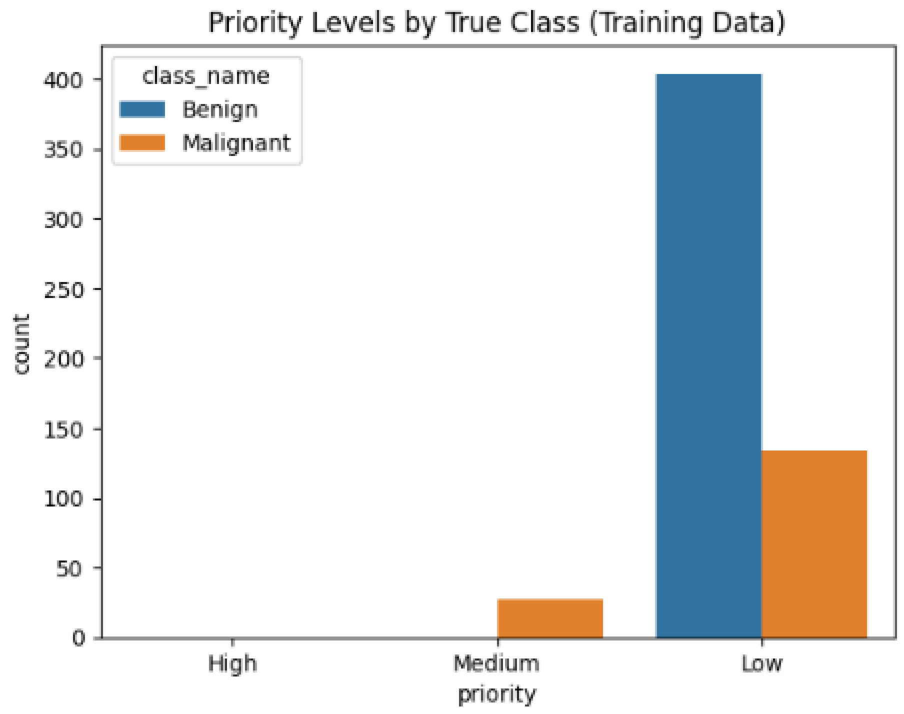
```
Accuracy: 0.74
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.88      0.84        85
           1       0.47      0.32      0.38        28

    accuracy                           0.74       113
   macro avg       0.64      0.60      0.61       113
weighted avg       0.72      0.74      0.73       113


Sample: {'mean': 114.80317914107493, 'std': 60.150502354807976, 'min': 0.0, 'max':
255.0}
Predicted Health: Healthy
Actual Health: Unhealthy
Predicted Value for second sample: 0.00
Actual Value: 0.00
```

The model is 74% accurate on the whole. It correctly identified benign cases 88% of the time but only 32% for malignant cases. This has been attributed to bias obtained from a larger number of benign samples in comparison to malignant samples availed.

Confusion matrix:



Bar plot:



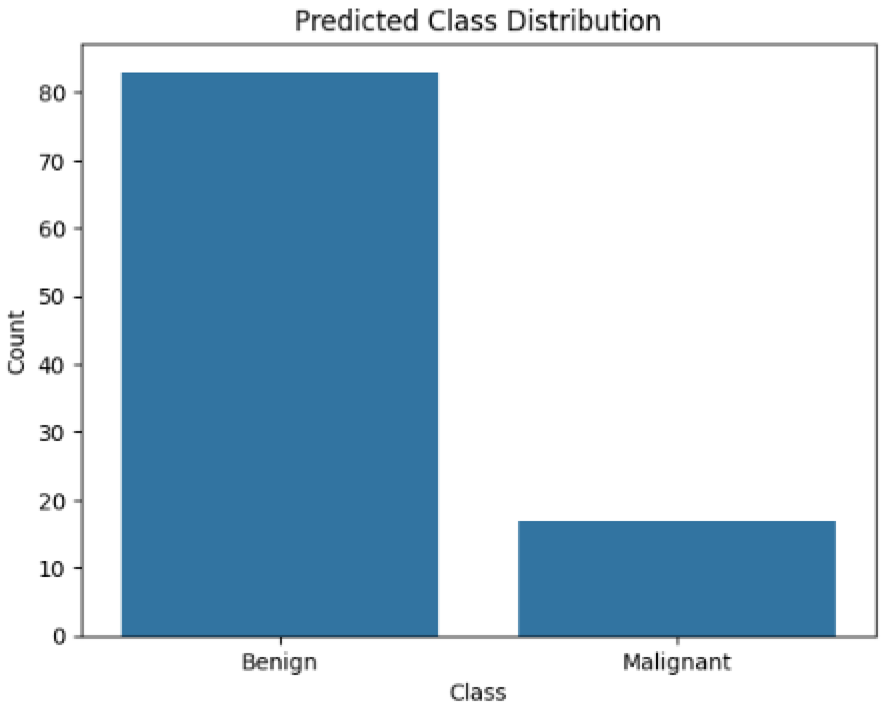
Testing Set:

```
  study_id                                         image_path  \
0     P025  /content/drive/MyDrive/complete_set/testing_se...
1     P011  /content/drive/MyDrive/complete_set/testing_se...
2     P004  /content/drive/MyDrive/complete_set/testing_se...
3     P021  /content/drive/MyDrive/complete_set/testing_se...
4     P034  /content/drive/MyDrive/complete_set/testing_se...
```

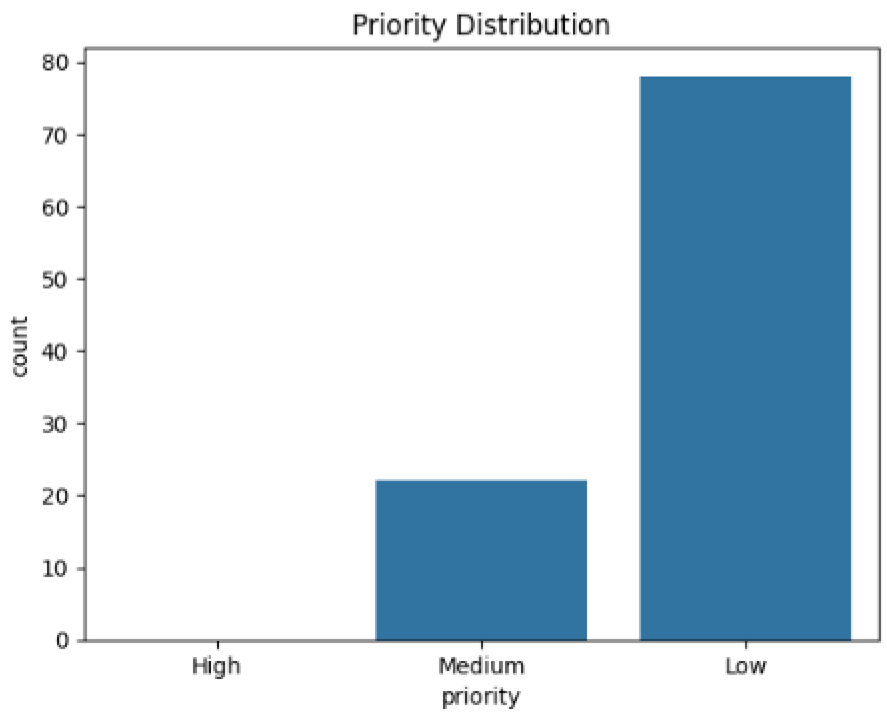
```
                                           mask_path  predicted_label  \
0  /content/drive/MyDrive/complete_set/testing_se...                0
1  /content/drive/MyDrive/complete_set/testing_se...                1
2  /content/drive/MyDrive/complete_set/testing_se...                0
3  /content/drive/MyDrive/complete_set/testing_se...                0
4  /content/drive/MyDrive/complete_set/testing_se...                0

  predicted_class  prob_malignant  prob_benign priority
0          Benign            0.24         0.76      Low
1       Malignant            0.67         0.33   Medium
2          Benign            0.25         0.75      Low
3          Benign            0.19         0.81      Low
4          Benign            0.38         0.62      Low
```

For unclassified test samples, they are classified with probability scores given as well. A priority is then assigned according to the weight of their probability and classification. A sample with a malignant probability score of 90% or above automatically becomes high priority case. It can also be high when the score is as low as 75% but the mean value of the pixels is above 150. Samples with a malignancy probability as low as 60% or those with a mean value above 120 pixels are classified as medium priority cases. Any sample below such is a low priority case.

Prediction count:



Priority distribution:



## Part 3

***Ethical Reflection and Fairness in Predictive Models***

When the predictive model is deployed in a company, potential biases can arise from the data used to train it.
For example, if some teams or departments are underrepresented in the dataset, the model may favor the groups with more data.
Historical or labeling biases can also occur if certain teams were often given higher priorities in the past, or if human judgments were inconsistent.
These biases can lead to unfair predictions, such as one team consistently receiving lower predicted priority than others.

Fairness tools like IBM AI Fairness 360 (AIF360) can help detect and reduce such biases.
The toolkit measures fairness using metrics like statistical parity and disparate impact to identify whether the model treats groups unequally.
It also provides bias mitigation methods before, during, or after training—such as balancing data, adjusting learning algorithms, or modifying predictions.
Using AIF360 helps make the model's decisions more transparent, fair, and trustworthy for all teams in the company.