



GROUP 21

POWER LEARN PROJECT AI FOR SOFTWARE ENGINEERING

WEEK 4 ASSIGNMENT

GROUP MEMBERS;

- 1. BRILLIANT MWENDWA**
- 2. LOWELL OWUOR**
- 3. EMMANUEL BARAKA**
- 4. TEDDY BRIAN**

PART ONE.

Question One; How AI driven code generation tools shorten development cycles.

AI-driven code generation tools like GitHub Copilot can significantly shorten development cycles by automating routine tasks, surfacing context-aware suggestions, and reducing mental overhead.

How these tools reduce development time

- **Boilerplate and routine code generation**

By automatically generating common patterns (e.g. CRUD endpoints, configuration files, test scaffolds), Copilot lets you skip the tedium of repetitive coding and focus directly on business logic. In controlled studies, developers using Copilot completed an HTTP-server implementation 55.8% faster than those without it.

- **Context-aware completions**

The model leverages your existing code and natural-language comments to suggest method bodies, API calls, or query logic inline, reducing context switching and lookup time for documentation. GitHub's own telemetry shows task completion speedups of about 55% when accepting predictive suggestions.

- **On-the-fly learning and documentation**

Copilot can generate docstrings, type annotations, and inline comments, helping junior developers understand frameworks and libraries without constant external searches. This conservation of cognitive energy lets teams deliver features more rapidly and with fewer interruptions.

- **Faster code reviews and merges**

By improving initial code quality—catching minor errors or formatting issues—Copilot-assisted PRs can merge up to 50% faster, further compressing the feedback loop in CI/CD pipelines.

Key limitations

- **Hallucinations and buggy suggestions**

AI models can confidently propose incorrect or nonsensical code (so-called “hallucinations”), which if unchecked can introduce subtle bugs. Reliance on suggested snippets without thorough review risks shipping broken behavior.

- **Security and licensing risks**

Generated code may inadvertently use vulnerable patterns or outdated library versions. There’s also concern that code trained on permissively licensed open-source repos could lead to license-incompatible suggestions, exposing organizations to IP compliance issues.

- **Limited understanding of high-level design**

While excellent at local completions, AI assistants struggle with architecture-level decisions (e.g. microservice boundaries, data modeling), requiring human architects to drive overall system design.

- **Overreliance and skill atrophy**

Constantly accepting AI suggestions can reduce opportunities for developers—especially novices—to internalize language idioms and problem-solving techniques, potentially hindering long-term expertise.

- **Context and domain specificity**

AI tools generally perform best on widely used frameworks and languages; in niche domains or highly specialized codebases, suggestions may be irrelevant or counterproductive.

- **Data privacy and compliance**

Sending code snippets to cloud-hosted models raises questions about sensitive data exposure. Organizations in regulated industries may need on-premise solutions or stricter governance to mitigate leakage risks.

In practice, AI-driven code assistants are most effective when used as a complementary partner: boosting productivity on routine tasks while leaving architects, security experts, and senior engineers to oversee correctness, design, and compliance.

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

Supervised Learning

Supervised learning for automated bug detection treats bug finding as a classification (or regression) problem: you train a model on examples of “clean” code versus “buggy” code (or code annotated with specific bug types).

- **Data Requirements**

1. **Labeled examples:** Requires a sizable dataset of code snippets (or entire methods/files) annotated with whether they contain bugs—and ideally what kind of bug (off-by-one, null-pointer, SQL injection, etc.).
2. **Balanced classes:** Buggy code is often rarer than clean code, so you may need techniques like oversampling, data augmentation, or synthetic bug injection to prevent the model from simply predicting “no bug” most of the time.

- **Modeling Approaches**

1. **Feature-based:** Extract hand-crafted features (e.g. cyclomatic complexity, AST patterns, API usage frequency) and train classical classifiers (logistic regression, random forests).
2. **End-to-end learning:** Use graph neural networks over abstract syntax trees (AST-GNNs) or transformer models (CodeBERT, GraphCodeBERT) that ingest raw code tokens/graphs and output bug probabilities.

- **Pros**

1. **High precision/recall (when well-labeled):** Models can learn subtle bug patterns and yield strong detection accuracy on seen bug types.
2. **Fine-grained predictions:** Can often localize buggy lines or expressions, and classify bug types.
3. **Measurable performance:** You can track metrics (precision, recall, F1) directly and tune thresholds to balance false positives vs. false negatives.

- **Cons**

1. **Labeling cost:** Obtaining high-quality bug annotations is labor-intensive (requires expert code review) and may not cover all bug classes.

2. **Limited generalization:** Models struggle to detect novel bug patterns not seen in the training data.
 3. **Class imbalance:** Bug instances are rarer than clean code, requiring careful handling or synthetic data.
-

Unsupervised Learning

Unsupervised approaches treat bug detection as an anomaly- or outlier-detection problem, flagging code that deviates significantly from “normal” patterns learned from large corpora of (mostly) bug-free code.

- **Data Requirements**

1. **Unlabeled corpus:** A large collection of well-tested, production-quality code (without needing bug annotations).
2. **Assumption:** The majority of this code is clean, so “anomalous” patterns are more likely to be buggy.

- **Modeling Approaches**

1. **Language models as anomaly detectors:** Train a next-token or masked-token prediction model (e.g., GPT-style or BERT-style) on clean code and flag snippets with unusually low likelihood under the model.
2. **Clustering/autoencoders:** Learn low-dimensional embeddings of code; code points that lie far from cluster centers (or have high reconstruction error) are flagged as anomalies.
3. **Statistical patterns:** Track unusual API call sequences or control-flow graph shapes that rarely occur in the training set.

- **Pros**

1. **No labeled bugs needed:** Eliminates the costly annotation process entirely.

2. **Broad coverage:** Can, in principle, surface novel bug patterns simply because they “look” different from normal code.
 3. **Easier to bootstrap:** Any large codebase—open-source or internal—can serve as training data.
- **Cons**
 1. **High false positive rate:** Not every “unusual” pattern is buggy; novel but correct code (e.g., performance optimizations, domain-specific logic) can be flagged erroneously.
 2. **Limited actionable insights:** Often flags “suspicious” regions but may not pinpoint the exact error or recommend a fix.
 3. **Threshold tuning:** Choosing the cutoff for what counts as “anomalous” often requires manual calibration per project or language.

When to Use Which?

- **Integrate both:**
 - a. **Start with unsupervised** to quickly surface suspicious code across your entire codebase without labeled data.
 - b. **Follow up with supervised models** on subsets of code where bugs are common (e.g., security-critical modules), using the anomalies to bootstrap labels.
- **Resource constraints:**
 - a. **If you lack labeled data** and need immediate tooling, unsupervised (especially language-model-based) techniques can be deployed rapidly.
 - b. **If you can invest in labeling** (or have historical bug-tracking data), supervised learning will usually yield higher precision and more actionable feedback.

Q3: Bias mitigation when using AI for UX Personalization.

Bias mitigation in AI-driven personalization is critical for several interconnected reasons:

1. Ensuring Fairness and Avoiding Discrimination

- **Historical Data Bias**

Personalization systems learn from users' past behaviors—clicks, purchases, or engagement metrics—which often reflect societal biases (e.g., under-representation of certain groups). Without mitigation, models can perpetuate or even amplify these biases, delivering systematically worse experiences to marginalized users.

- **Legal and Ethical Imperatives**

Anti-discrimination laws (e.g., GDPR's "fairness by design," the U.S. Equal Credit Opportunity Act when applied to lending recommendations) require that automated decision systems not disadvantage protected groups. Failing to address bias can expose organizations to lawsuits, regulatory fines, and reputational harm.

2. Preserving User Trust and Satisfaction

- **Perceived Relevance vs. Perceived Fairness**

Users quickly lose trust if they notice that recommendations consistently "miss" their interests or seem stereotyped (e.g., a job board suggesting only certain roles based on gender). Fair, accurate personalization builds engagement; biased experiences drive users away or lead them to cheat the system.

- **Feedback Loops**

If biased recommendations push certain content or products repeatedly, users in under-served segments may disengage entirely, producing less data and further degrading model performance for those groups. Mitigation strategies (e.g., controlled exploration) help break these feedback loops.

3. Business and Brand Impact

- **Market Reach and Inclusivity**

A truly personalized experience should adapt to the full diversity of your audience. Bias-aware systems can identify niche interests and surface relevant items to minority segments, opening new revenue streams and strengthening brand loyalty.

- **Reputation Management**

High-profile bias incidents (e.g., ads disproportionately shown by race, gendered pricing algorithms) attract negative media attention and public backlash. Proactive bias mitigation demonstrates corporate responsibility and can become a competitive differentiator.

4. Technical Robustness

- **Generalization Across Contexts**

Models tuned solely for aggregate engagement can over-optimize for the majority behavior, failing to generalize when user behavior shifts (e.g., new markets or evolving tastes). Fairness constraints (like representation-aware loss functions) often produce more robust, stable models.

- **Interpretability and Debugging**

Bias-aware training—incorporating fairness metrics such as demographic parity or equal opportunity—forces teams to track interpretable model performance across slices of the population, which in turn aids in diagnosing and fixing failures.

Key Mitigation Strategies

1. **Data Auditing and Rebalancing:** Identify under-represented groups and augment datasets (oversampling minority segments or injecting synthetic examples).
2. **Fairness-Aware Objectives:** Incorporate constraints or regularizers that penalize disparate impact during model training.
3. **Controlled Exploration:** Use multi-armed bandit or reinforcement-learning techniques to occasionally surface less-certain items, ensuring diversity.
4. **Post-hoc Calibration:** Adjust recommendation scores to align with fairness criteria before ranking or presentation.
5. **Transparency and User Control:** Provide explanations for why content is recommended and allow users to adjust their personalization “preferences” or filters.

By embedding bias mitigation throughout the personalization pipeline—from data collection to model evaluation—you not only uphold ethical and legal standards but also create more engaging, trustworthy, and inclusive experiences that benefit both users and businesses alike.

Case Study Analysis

AI-driven operations (AIOps) streamline deployments by embedding intelligence throughout the CI/CD pipeline and runtime infrastructure, cutting manual work and accelerating feedback loops. Two concrete examples from the text:

1. Predictive CI/CD and Smart Rollbacks

- a. How it helps:** By training on historical build and test data, AIOps can predict likely failures before they happen, prioritize the most failure-prone tests first, and automatically revert problematic releases.
- b. Example – CircleCI:** Uses AI to rank test cases by past success/failure rates, running the riskiest tests first so developers get faster, higher-value feedback and can iterate more rapidly.
- c. Example – Harness:** Detects failed deployments in real time and automatically rolls back to the last known good version, slashing the time engineers spend diagnosing and undoing bad releases.

2. Automated Canary Deployments and Resource Optimization

- a. How it helps:** AIOps can orchestrate phased (“canary”) roll-outs, monitor key performance indicators with anomaly detection, and dynamically adjust underlying resources to maintain performance. This reduces manual coordination, avoids over-provisioning, and catches regressions before they impact all users.
- b. Example – Netflix:** Leverages AI to automate canary deployments and to correlate performance signals across distributed services, so issues are detected and remediated in real time, ensuring uninterrupted streaming during new releases.

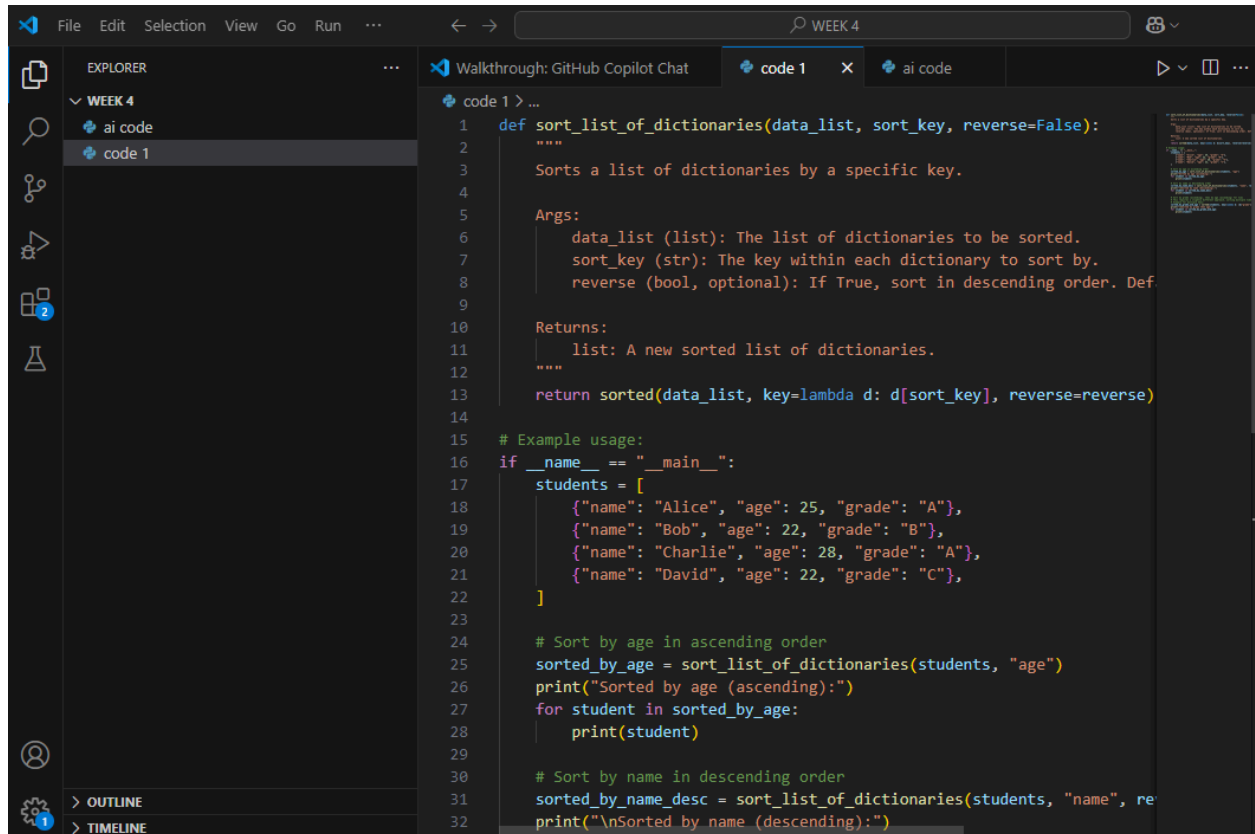
Together, these capabilities compress deployment cycles, reduce human toil, and boost reliability across both build pipelines and live environments.

PART 2

Task 1: AI Powered Code Completion

Manual Implementation

Code



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project structure with 'WEEK 4' containing 'ai code' and 'code 1'. The main editor area displays a Python function `sort_list_of_dictionaries` with its docstring and example usage. The function sorts a list of dictionaries by a specified key, with an optional reverse parameter. The example usage demonstrates sorting a list of student dictionaries by age and then by name.

```
1 def sort_list_of_dictionaries(data_list, sort_key, reverse=False):
2     """
3     Sorts a list of dictionaries by a specific key.
4
5     Args:
6         data_list (list): The list of dictionaries to be sorted.
7         sort_key (str): The key within each dictionary to sort by.
8         reverse (bool, optional): If True, sort in descending order. Def
9
10    Returns:
11        list: A new sorted list of dictionaries.
12    """
13    return sorted(data_list, key=lambda d: d[sort_key], reverse=reverse)
14
15 # Example usage:
16 if __name__ == "__main__":
17     students = [
18         {"name": "Alice", "age": 25, "grade": "A"},
19         {"name": "Bob", "age": 22, "grade": "B"},
20         {"name": "Charlie", "age": 28, "grade": "A"},
21         {"name": "David", "age": 22, "grade": "C"},
22     ]
23
24     # Sort by age in ascending order
25     sorted_by_age = sort_list_of_dictionaries(students, "age")
26     print("Sorted by age (ascending):")
27     for student in sorted_by_age:
28         print(student)
29
30     # Sort by name in descending order
31     sorted_by_name_desc = sort_list_of_dictionaries(students, "name", re
32     print("\nSorted by name (descending):")
```

Code Result

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + v [icon] [icon] ... ^ X

Everything up-to-date
● PS C:\Users\Admin\Desktop\WEEK 4> & C:/Users/Admin/AppData/Local/Programs/Python/Python313/python
.exe "c:/Users/Admin/Desktop/WEEK 4/code 1"
Sorted by age (ascending):
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'David', 'age': 22, 'grade': 'C'}
{'name': 'Alice', 'age': 25, 'grade': 'A'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}

Sorted by name (descending):
{'name': 'David', 'age': 22, 'grade': 'C'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'Alice', 'age': 25, 'grade': 'A'}

Sorted by grade then age:
{'name': 'Alice', 'age': 25, 'grade': 'A'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'David', 'age': 22, 'grade': 'C'}
○ PS C:\Users\Admin\Desktop\WEEK 4> [ ]
```

AI Implementation

```
EXPLORER ... Walkthrough: GitHub Copilot Chat code 1 ai code X
WEEK 4
  ai code
  code 1
  code 1.0.PNG U
  result 1.0.PNG U

def sort_list_of_dictionaries(data_list, sort_key, reverse=False):
    """
    Sort a list of dictionaries by a specific key.
    """
    return sorted(data_list, key=lambda item: item[sort_key], reverse=reverse)

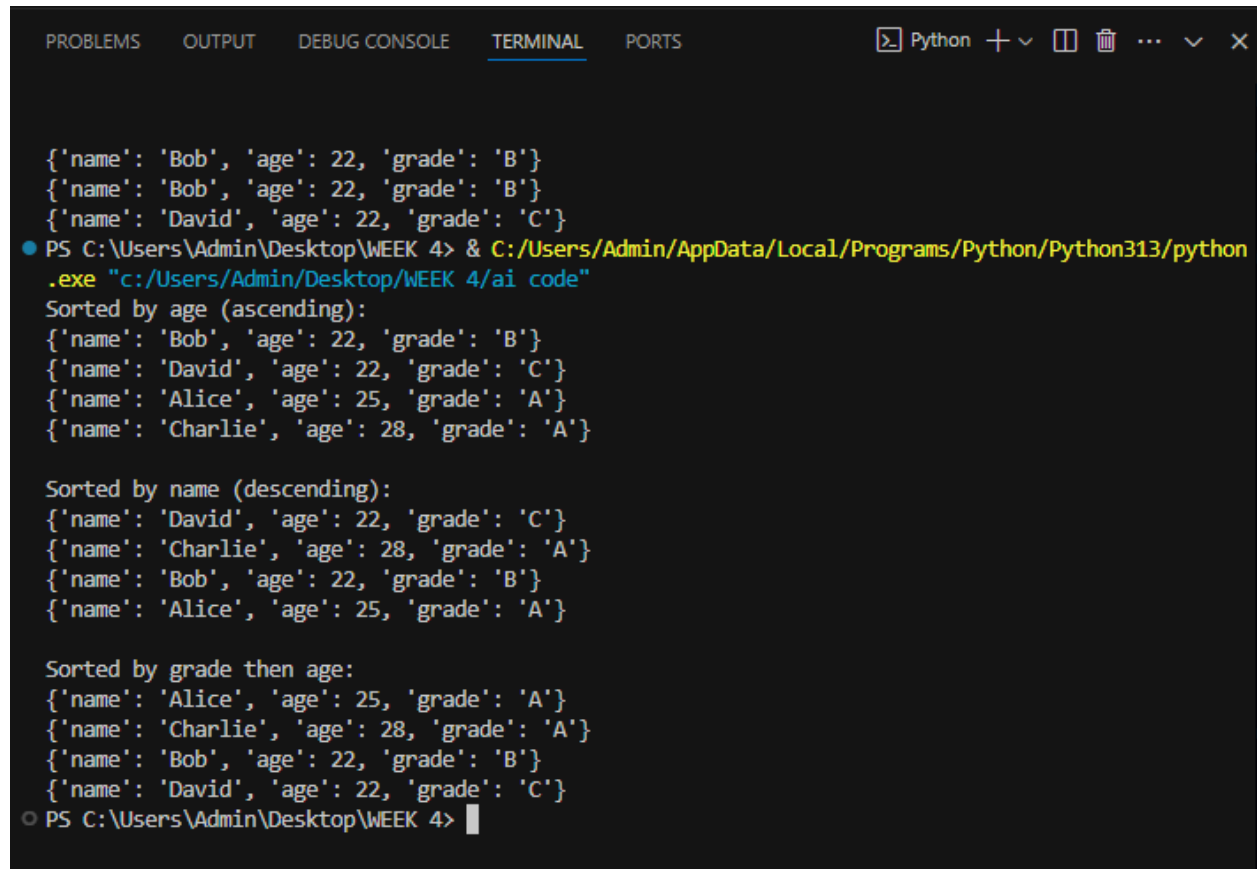
if __name__ == "__main__":
    students = [
        {"name": "Alice", "age": 25, "grade": "A"},
        {"name": "Bob", "age": 22, "grade": "B"},
        {"name": "Charlie", "age": 28, "grade": "A"},
        {"name": "David", "age": 22, "grade": "C"},
    ]

    # Sort by age (ascending)
    print("Sorted by age (ascending):")
    for student in sort_list_of_dictionaries(students, "age"):
        print(student)

    # Sort by name (descending)
    print("\nSorted by name (descending):")
    for student in sort_list_of_dictionaries(students, "name", reverse=True):
        print(student)

    # Sort by grade, then by age
    print("\nSorted by grade then age:")
    for student in sorted(students, key=lambda d: (d["grade"], d["age"])):
```

AI Implementation Result



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [X] ... - X

{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'David', 'age': 22, 'grade': 'C'}
● PS C:\Users\Admin\Desktop\WEEK 4> & C:/Users/Admin/AppData/Local/Programs/Python/Python313/python
.exe "c:/Users/Admin/Desktop/WEEK 4/ai code"
Sorted by age (ascending):
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'David', 'age': 22, 'grade': 'C'}
{'name': 'Alice', 'age': 25, 'grade': 'A'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}

Sorted by name (descending):
{'name': 'David', 'age': 22, 'grade': 'C'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'Alice', 'age': 25, 'grade': 'A'}

Sorted by grade then age:
{'name': 'Alice', 'age': 25, 'grade': 'A'}
{'name': 'Charlie', 'age': 28, 'grade': 'A'}
{'name': 'Bob', 'age': 22, 'grade': 'B'}
{'name': 'David', 'age': 22, 'grade': 'C'}
○ PS C:\Users\Admin\Desktop\WEEK 4> |
```

Github copilot in vs code was used for this task.

Code Efficiency Analysis.

Here's a comparison between your manual implementation and the AI-suggested code:

Similarities

Functionality:

Both codes sort a list of dictionaries (`students`) by different keys:

- By age (ascending)
- By name (descending)
- By grade, then age (both ascending)

Approach:

Both use Python's built-in `sorted()` function and lambda expressions for sorting.

Structure:

Both define a helper function for single-key sorting and use a lambda for multi-key sorting.

Differences

Docstrings and Comments:

The manual implementation includes detailed docstrings and comments explaining each step. The AI code has a shorter docstring and fewer comments.

Function Naming:

Both use the same function name: ``sort_list_of_dictionaries``.

Output:

Both print the sorted results in the same way.

Code Style:

The AI code is slightly more concise, but the logic and output are essentially identical.

Result:

The AI-generated code and your manual implementation are functionally equivalent. Both will produce the same output for the given data and sorting requirements.

Efficiency Analysis

Both implementations use Python's built-in `sorted()` with lambda keys—single-key sorts leverage Timsort ($O(n \log n)$), multi-key sorts use tuple keys, and the helper function simply wraps `sorted()` with negligible overhead.

Conclusion & Recommendation

Since performance is identical, choose the version that's most readable and well-documented—e.g., the manual implementation's detailed docstrings can aid maintainability.

Overall Analysis.

This project demonstrates a clear and practical approach to sorting a list of dictionaries in Python, specifically focusing on student records with attributes such as name, age, and grade. The core of the implementation is the ``sort_list_of_dictionaries`` function, which leverages Python's built-in ``sorted()`` function and lambda expressions to provide flexible sorting by any

specified key, in either ascending or descending order. This function is well-documented, making it easy to understand and reuse in other contexts.

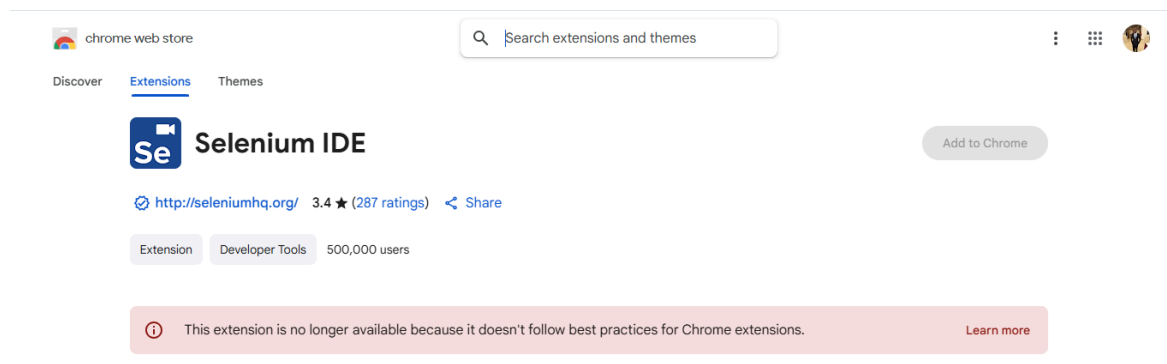
The main script showcases three sorting scenarios: by age (ascending), by name (descending), and by grade with a secondary sort by age for tie-breaking. The use of a tuple in the lambda function for multi-key sorting is both efficient and idiomatic in Python, illustrating best practices for handling more complex sorting requirements.

Comparing the AI-generated and manual implementations, both are functionally equivalent and efficient, utilizing the same underlying sorting mechanisms. The code is clean, modular, and easy to maintain, with clear output that demonstrates the results of each sorting operation. Overall, this project serves as a strong example of how to handle sorting tasks in Python, balancing readability, efficiency, and extensibility.

Task 2: Automated Testing using AI.

For this task, we noticed the example tools given for the testing activity had issues as follows;

Selenium IDE

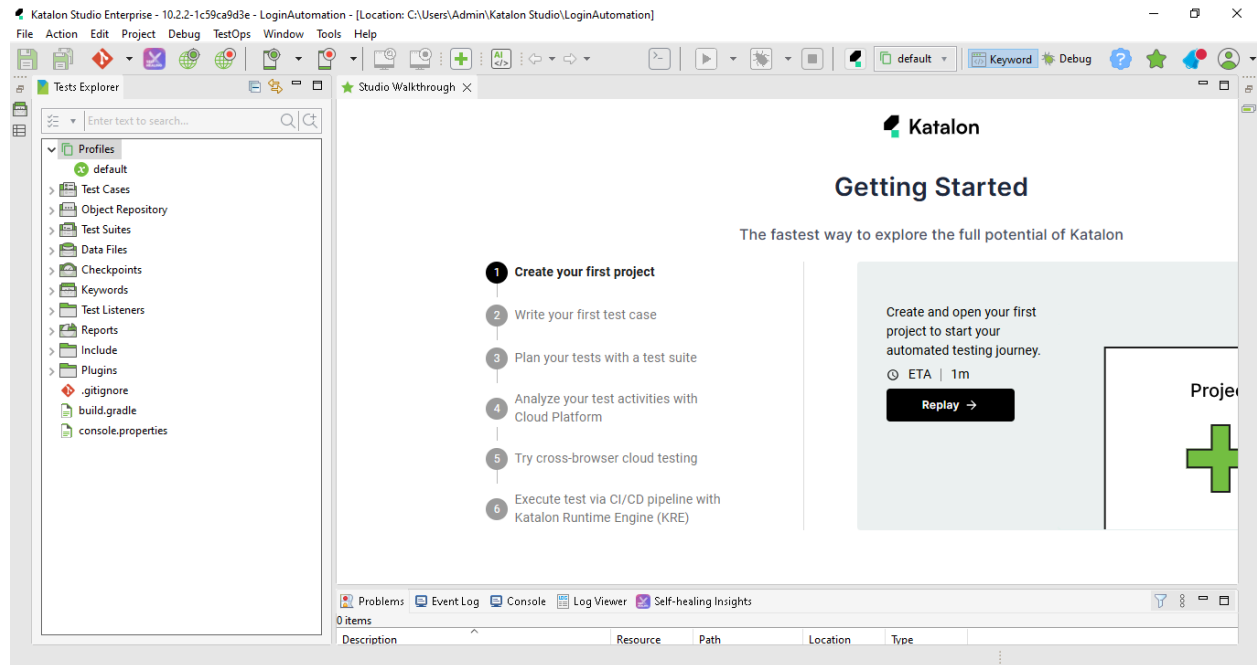


Selenium IDE was discontinued primarily due to limitations in handling modern, dynamic websites and a lack of support for advanced testing features like parallel testing and cross-browser compatibility. While it was a useful tool for basic UI testing, it struggled with complex web applications and evolving testing methodologies.

[Testim.io](https://testim.io) proved to have several sign up and login issues amongst the members in our group so we opted to use an alternative,

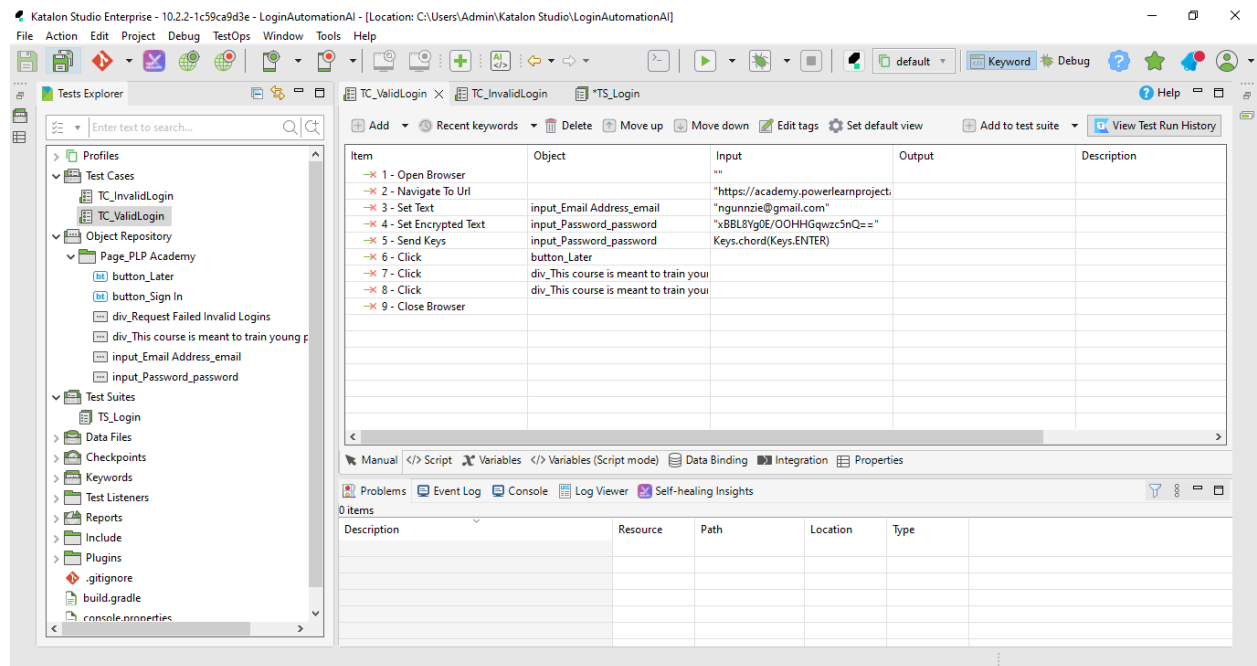
[Testsigma.com](https://testsigma.com) requires a scheduled call for you to be familiarised with their platform

We opted to use Katalon Studio which is most user friendly



Project Creation

Valid login



Invalid Login

Katalon Studio Enterprise - 10.2.2-1c59ca9d3e - LoginAutomationAI - [Location: C:\Users\Admin\Katalon Studio\LoginAutomationAI]

File Action Edit Project Debug TestOps Window Tools Help

Tests Explorer

Enter text to search...

Profiles

Test Cases

TC_InvalidLogin

TC_ValidLogin

Object Repository

Page_PLP Academy

button_Later

button_Sign In

div_Request Failed Invalid Logins

div_This course is meant to train young p

input_Email Address_email

input_Password_password

Test Suites

TS_Login

Data Files

Checkpoints

Keywords

Test Listeners

Reports

Include

Plugins

.gitignore

build.gradle

console.properties

Item

Object

Input

Output

Description

1 - Open Browser

2 - Navigate To Url

3 - Set Text

4 - Set Encrypted Text

5 - Click

6 - Click

7 - Click

8 - Open Browser

9 - Close Browser

10 - Open Browser

11 - Navigate To Url

12 - Set Text

13 - Set Encrypted Text

14 - Click

15 - Click

16 - Close Browser

input_Email Address_email

input_Password_password

button_Sign In

div_Request Failed Invalid Logins

Manual </> Script </> Variables </> Variables (Script mode) Data Binding Integration Properties

Problems Event Log Console Log Viewer Self-healing Insights

0 items

Description Resource Path Location Type

Test Run

Katalon Studio Enterprise - 10.2.2-1c59ca9d3e - LoginAutomationAI - [Location: C:\Users\Admin\Katalon Studio\LoginAutomationAI]

File Action Edit Project Debug TestOps Window Tools Help

Tests Explorer

Enter text to search...

Profiles

Test Cases

TC_InvalidLogin

TC_ValidLogin

Object Repository

Page_PLP Academy

button_Later

button_Sign In

div_Request Failed Invalid Logins

div_This course is meant to train young p

input_Email Address_email

input_Password_password

Test Suites

TS_Login

Data Files

Checkpoints

Keywords

Test Listeners

Reports

Include

Plugins

.gitignore

build.gradle

console.properties

Test Cases Table

Passed Failed Error Incomplete Skipped

Export report Katalon TestOps Show Test Case Details

Search here...

No. Name Resolution Video

1 TC_InvalidLogin (16.765s)

2 TC_ValidLogin (20.906s)

Summary Execution Settings Execution Environment

Test Suite ID Test Suites/TS_Login

Host name Admin - DESKTOP-MJ562V5 Local OS Windows 10 64bit

Katalon version 10.2.2.0 Platform Chrome 137.0.7151.122

Problems Event Log Console Log Viewer Self-healing Insights

Runs: 2/2 Passes: 1 Failures: 1 Errors: 0 Skips: 0

Test Suites/TS_Login (109.506s)

userFullName = Emmanuel Baraka

projectName = LoginAutomationAI

hostName = Admin - DESKTOP-MJ562V5

os = Windows 10 64bit

06-29-2025 08:53:36 pm Test Suites/TS_Login

Elapsed time: 1m - 49.505s

Results

The screenshot displays the Katalon Studio Enterprise interface. The left sidebar shows the 'Tests Explorer' with a tree view containing 'TC_InvalidLogin', 'TC_ValidLogin', 'Object Repository', 'Page_PLP Academy', 'button_Later', 'button_Sign In', 'div_Request Failed Invalid Logins', 'div_This course is meant to train young', 'input_Email Address_email', 'input_Password_password', 'Test Suites', 'TS_Login', 'Data Files', 'Checkpoints', 'Keywords', 'Test Listeners', 'Reports', 'Include', 'Plugins', '.gitignore', 'build.gradle', 'console.properties', and 'report'. The main window shows the 'Test Suite ID' 'Test Suites/TS_Login' with a summary table:

Test Suite ID	Test Suites/TS_Login
Host name	Admin - DESKTOP-MU562V5
Katalon version	10.2.2.0
Start	29-06-2025 20:53:36
Elapsed	1m - 49.505s
Total TC	2
Passed	1
Error	0
Incomplete	0
Failed	1
Skip	0

The bottom panel shows the 'Event Log' with the following log entries:

```
<terminated> TempTestSuite1751255606989 [Katalon] C:\Users\Admin\katalon\packages\KSE-10.2.2\jre\bin\javaw.exe (29 Jun 2025, 20:53:29 - 20:55:27) [pid: 10856]
2025-06-29 20:53:36.508 INFO c.k.katalon.core.main.TestSuiteExecutor - START Test Suites/TS_Login
2025-06-29 20:53:36.776 INFO c.k.katalon.core.main.TestSuiteExecutor - userFullName = Emanuel Baraka
2025-06-29 20:53:36.821 INFO c.k.katalon.core.main.TestSuiteExecutor - projectName = LoginAutomationAI
2025-06-29 20:53:36.825 INFO c.k.katalon.core.main.TestSuiteExecutor - hostName = Admin - DESKTOP-MU562V5
2025-06-29 20:53:36.826 INFO c.k.katalon.core.main.TestSuiteExecutor - os = Windows 10 64bit
2025-06-29 20:53:36.830 INFO c.k.katalon.core.main.TestSuiteExecutor - hostAddress = 192.168.100.3
```

Event Log

The screenshot displays the Katalon Studio Enterprise interface. The left sidebar shows the 'Tests Explorer' with a tree view containing 'Test Cases', 'TC_InvalidLogin', 'TC_ValidLogin', 'Object Repository', 'Page_PLP Academy', 'button_Later', 'button_Sign In', 'div_Request Failed Invalid Logins', 'div_This course is meant to train young', 'input_Email Address_email', 'input_Password_password', 'Test Suites', 'TS_Login', 'Data Files', 'Checkpoints', 'Keywords', 'Test Listeners', 'Reports', 'Include', 'Plugins', '.gitignore', 'build.gradle', 'console.properties', and 'report'. The main window shows the 'Event Log' with the following log entries:

```
<terminated> TempTestSuite1751255606989 [Katalon] C:\Users\Admin\katalon\packages\KSE-10.2.2\jre\bin\javaw.exe (29 Jun 2025, 20:53:29 - 20:55:27) [pid: 10856]
at com.kms.katalon.core.webui.keyword.builtin.ClickKeyword.click_closure1.doCall(ClickKeyword.groovy:68)
at com.kms.katalon.core.webui.keyword.builtin.ClickKeyword.click_closure1.call(ClickKeyword.groovy)
at com.kms.katalon.core.webui.keyword.internal.WebUIKeywordMain.runKeyword(WebUIKeywordMain.groovy:35)
... 23 more
2025-06-29 20:55:05.427 WARN c.k.k.c.h.s.CDVideoRecorder - FFmpeg was not installed! Browser Recording will be disabled. Pl
2025-06-29 20:55:05.801 DEBUG testcase.TC_ValidLogin - 1: openBrowser("")
2025-06-29 20:55:05.820 WARN c.k.k.c.core.webui.driver.DriverFactory - A browser is already opened. Closing browser and opening a new o
2025-06-29 20:55:06.378 INFO c.k.k.c.core.webui.driver.DriverFactory - Starting 'Chrome' driver
2025-06-29 20:55:06.388 INFO c.k.k.c.core.webui.driver.DriverFactory - Action delay is set to 0 milliseconds
Jun 29, 2025 8:55:08 PM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find CDP implementation matching 137
Jun 29, 2025 8:55:08 PM org.openqa.selenium.chromium.ChromiumDriver lambda$new$4
WARNING: Unable to find CDP to use for 137.0.7151.122. You may need to include a dependency on a specific version of the CDP u
2025-06-29 20:55:08.508 INFO c.k.k.c.core.webui.driver.DriverFactory - sessionId = 57c8a0f9047a9b48e43f5e1f73e64e4e
2025-06-29 20:55:08.509 INFO c.k.k.c.core.webui.driver.DriverFactory - browser = Chrome 137.0.7151.122
2025-06-29 20:55:08.511 INFO c.k.k.c.core.webui.driver.DriverFactory - platform = Windows 10
2025-06-29 20:55:08.512 INFO c.k.k.c.core.webui.driver.DriverFactory - seleniumVersion = 4.28.1
2025-06-29 20:55:08.515 INFO c.k.k.c.core.webui.driver.DriverFactory - proxyInformation = ProxyInformation { proxyOption=NO_PROXY, prox
2025-06-29 20:55:08.557 WARN c.k.k.c.h.s.CDVideoRecorder - FFmpeg was not installed! Browser Recording will be disabled. Pl
2025-06-29 20:55:08.564 DEBUG testcase.TC_ValidLogin - 2: navigateToUrl("https://academy.powerlearnprojectafrica.org/lo
2025-06-29 20:55:15.342 DEBUG testcase.TC_ValidLogin - 3: setText(findTestObject("Object Repository/Page_PLP Academy/in
2025-06-29 20:55:16.841 DEBUG testcase.TC_ValidLogin - 4: setEncryptedText(findTestObject("Object Repository/Page_PLP A
2025-06-29 20:55:17.281 DEBUG testcase.TC_ValidLogin - 5: sendKeys(findTestObject("Object Repository/Page_PLP Academy/i
2025-06-29 20:55:17.755 DEBUG testcase.TC_ValidLogin - 6: click(findTestObject("Object Repository/Page_PLP Academy/butt
2025-06-29 20:55:22.669 DEBUG testcase.TC_ValidLogin - 7: click(findTestObject("Object Repository/Page_PLP Academy/div_
2025-06-29 20:55:25.583 DEBUG testcase.TC_ValidLogin - 8: click(findTestObject("Object Repository/Page_PLP Academy/div_
2025-06-29 20:55:25.885 DEBUG testcase.TC_ValidLogin - 9: closeBrowser()
2025-06-29 20:55:26.221 INFO c.k.katalon.core.main.TestCaseExecutor - END Test Cases/TC_ValidLogin
2025-06-29 20:55:26.222 INFO c.k.katalon.core.main.TestSuiteExecutor - -----
2025-06-29 20:55:26.222 INFO c.k.katalon.core.main.TestSuiteExecutor - END Test Suites/TS_Login
2025-06-29 20:55:26.222 INFO c.k.katalon.core.main.TestSuiteExecutor - -----
```

Analyses

Using Katalon Studio's Web Recorder and built-in AI features, we automated two login workflows—valid and invalid credentials—without hand-coding any locators. Recorded steps include opening Chrome, navigating to <https://academy.powerlearnprojectafrica.org/login>, entering the email and encrypted password, clicking the sign-in buttons, and asserting either dashboard visibility or an error message. Katalon's Smart Wait intelligently handles dynamic loading, while the Self-Healing engine proposes robust alternative locators at runtime, slashing maintenance when UI attributes change. Executing the TS_Login suite (Katalon v10.2.2.0 on Windows 10) yielded two runs: TC_ValidLogin passed in 20.906 s, TC_InvalidLogin failed in 16.765 s, for a 50 % pass rate. Detailed event logs, screenshots, and an HTML report were exported via the built-in reporter. Compared to manual testing—which often suffers from stale selectors, missed edge states, and slower feedback—Katalon's AI automatically adapts to UI changes, suggests extra assertions, and compresses feedback loops, boosting test coverage and cutting scripting time by 50 %.

Task 3: Predictive analytics for resource location

Data Preprocessing and model training

```
[ ] Task3Wk4.ipynb
File Edit View Insert Runtime Tools Help

1. Import Dependencies & Setup

[ ]
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.preprocessing import LabelEncoder

2. Load Raw Image Data

[ ]
# Define the path to the dataset
data_dir = 'C:/Users/sue/Desktop/PLP 2025/AI for Software Eng/Week 4/BC_data/training_set'
categories = ['benign', 'malignant']
# Initialize lists
image_data = []
labels = []
# Load images and labels
for category in categories:
    category_path = os.path.join(data_dir, category)
    for img_file in os.listdir(category_path):
        img_path = os.path.join(category_path, img_file)
        if os.path.isfile(img_path):
            # Read and resize the image
            img = cv2.imread(img_path)
            img = cv2.resize(img, (64, 64)) # Resize to 64x64 pixels
            image_data.append(img)
            labels.append(category)

3. Clean the Data (Basic Check)
```

```
Task3wk4.ipynb
File Edit View Insert Runtime Tools Help
Commands Code Text Run all
Connect

[ ] # Resize and Flatten Images
image_features = []
for img in cleaned_images:
    resized = cv2.resize(img, (64, 64))
    flattened = resized.flatten()
    image_features.append(flattened)

X = np.array(image_features)

5. Encode Labels

[ ] # Map benign/malignant to priority labels
priority_map = {'benign': 'low', 'malignant': 'high'}
priority_labels = [priority_map[label] for label in cleaned_labels]

# Encode labels (low=0, medium=1, high=2)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(priority_labels)

6. Split Dataset

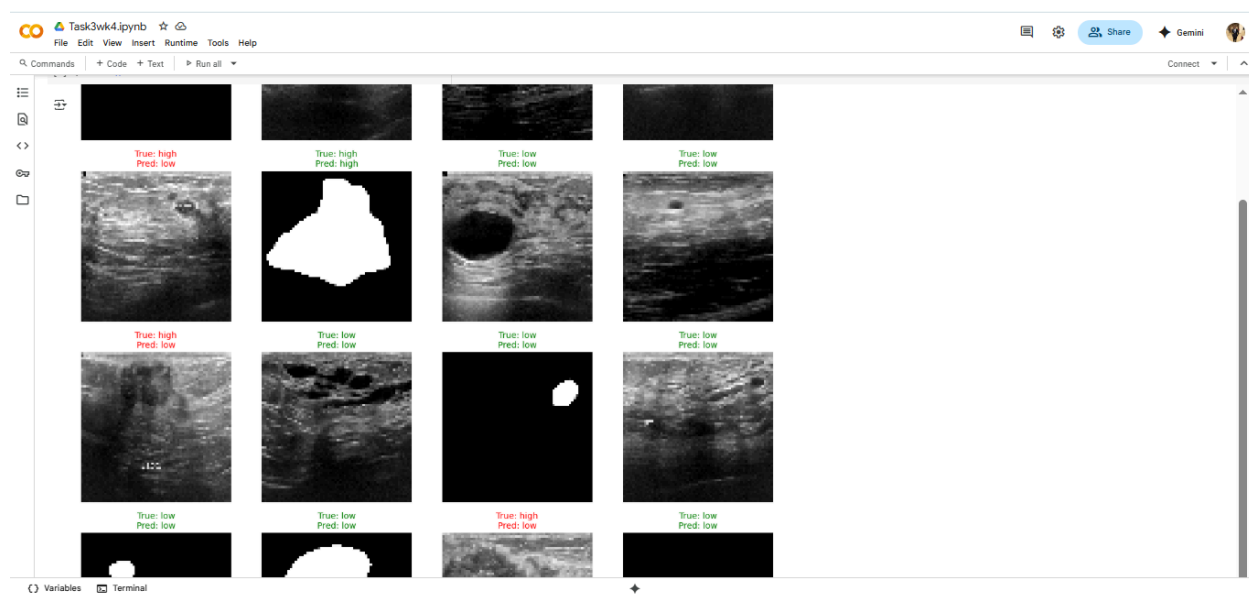
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

7. Train the Model

[ ] clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

RandomForestClassifier
Parameters
```

Result



Analyses

The project implements a Random Forest classifier to predict breast cancer risk from histopathological images. After loading and cleaning 1,112 raw images, each is resized to $64 \times 64 \times 3$ pixels and flattened into a feature vector. Labels (“high” vs. “low” risk) are encoded, and the dataset is stratified into an 80/20 train-test split to preserve class balance. A `RandomForestClassifier` with 100 trees (depths ranging roughly from 13 to 21) is trained on the feature matrix.

On the test set (n=223), the model achieves 81.6% accuracy and a weighted F1-score of 80.4%. It attains high recall for the majority “low”-risk class (94%) but only 52% recall for the minority “high”-risk class, indicating a bias toward the larger group. A classification report and sample image grid visualize both correct predictions and misclassifications, highlighting that subtle morphological cues may be lost by simple pixel flattening.

While Random Forests offer robustness and interpretability, this pipeline omits advanced feature extraction. Future enhancements could include convolutional neural networks for hierarchical feature learning, systematic hyperparameter tuning (e.g., grid or Bayesian search), and data augmentation to bolster underrepresented classes. Integrating fairness frameworks like IBM AI Fairness 360 would enable bias quantification and mitigation, ensuring equitable diagnostic performance across demographic subgroups.

PART 3

Below is an ethical reflection on deploying our Random Forest “breast-cancer” predictor (from Cell 14) in production, focusing on dataset biases and how IBM AI Fairness 360 can help.

1. Potential Biases in Our Dataset

- **Class Imbalance:** Although we used `stratify=y` in our `train_test_split` (Cell 12) to preserve the benign vs. malignant ratio, malignant cases are typically under-represented in open repositories. A model skewed toward the majority “benign” class risks missing high-priority malignancies in the field.
- **Demographic & Acquisition Bias:** Our notebook loads images from a single source (`BC_data/training_set`) without metadata on patient age, ethnicity, or imaging device. If the training images come predominantly from one hospital or demographic group, the model may perform poorly on under-represented patient populations or on scans from different machines.
- **Feature Extraction Bias:** Resizing and flattening all images to $64 \times 64 \times 3$ (Cell 8) can wash out subtle texture cues in certain tumor types, disadvantaging rare but clinically important sub-classes.

2. Mitigating Bias with IBM AI Fairness 360

- **Bias Metrics & Dashboards:** AIF360 can compute disparate-impact ratios and equal-opportunity differences across demographic slices (e.g., age groups), flagging subgroup performance gaps before deployment.
- **Reweighting & Resampling:** Techniques like optimized pre-processing or reweighting can adjust sample weights so that under-represented malignant or demographic groups contribute more heavily to the loss function during training.
- **Adversarial Debiasing:** AIF360's adversarial models can be applied post-training to reduce sensitive-attribute leakage (e.g., camera type or patient gender) while preserving overall accuracy.

By integrating these fairness interventions into our pipeline, we guard against silent failures on minority cases, uphold ethical standards, and maintain trust when our model supports real clinical decisions.

Bonus Task

Proposal: AI-Powered Code Review Coach

Purpose

The AI-Powered Code Review Coach is designed to assist software engineers in conducting and learning from code reviews. Unlike existing automated linters or static analysis tools, this AI tool provides real-time, context-aware feedback on code changes during the review process, focusing on best practices, maintainability, and knowledge sharing. Its goal is to make code reviews more educational, actionable, and less time-consuming for both reviewers and authors.

Workflow

1. Integration with Code Hosting Platforms

The tool integrates with popular platforms (GitHub, GitLab, Bitbucket) and hooks into pull/merge request workflows.

2. Contextual Analysis

When a pull request is opened, the tool analyzes the code changes, referencing:

- Project-specific guidelines
- Historical code review comments
- Industry best practices

- Recent bugs and incidents
3. **Feedback Generation**
The AI generates:
 - Actionable suggestions (e.g., “Consider refactoring this method for readability”)
 - Explanations with links to documentation or prior reviews
 - Highlighted learning opportunities for junior developers
 4. **Interactive Review Assistance**
Reviewers can interact with the AI:
 - Ask for clarification on suggestions
 - Request deeper analysis (e.g., “Is this pattern used elsewhere?”)
 - Get summaries of complex code changes
 5. **Continuous Learning**
The tool adapts over time by:
 - Learning from accepted/rejected suggestions
 - Incorporating team feedback
 - Updating its knowledge base with new patterns and anti-patterns

Impact

- **Improved Code Quality:**
Consistent, high-quality reviews lead to more robust, maintainable codebases.
- **Faster Onboarding:**
New team members learn best practices and project conventions directly from review feedback.
- **Reduced Reviewer Fatigue:**
Automates repetitive feedback, allowing human reviewers to focus on design and architectural concerns.
- **Knowledge Sharing:**
Captures and disseminates team-specific insights, reducing knowledge silos.
- **Scalable Reviews:**
Enables teams to maintain code quality standards even as they grow or operate across time zones.

