PLP AI in Software Engineering Week 4 Assignment /Report

Part 1:Theoretical Understanding

Q1: AI-Driven Code Generation

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

Answer: AI-driven tools significantly reduce development time by acting as intelligent pair programmers. They swiftly autocomplete boilerplate code, suggest entire functions based on context (docstrings and comments), and handle repetitive coding tasks. This minimizes context switching and allows developers to dedicate more focus to complex architectural logic.

Limitations include context blindness (difficulty understanding large, multi-file architectures), generating code that can be non-optimal, inefficient, or vulnerable to security flaws, and potential intellectual property (IP) and licensing concerns derived from their training data.

Q2:Supervised vs. Unsupervised Bug Detection

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

Answer: Supervised learning is trained on labeled data (where instances are explicitly marked as 'bug' or 'no bug'). It excels at identifying known bug patterns or deviations from established coding rules, as it learns direct input-output associations.

Unsupervised learning works on unlabeled data by identifying anomalies or clusters that deviate significantly from typical system behavior or code structure. This approach is more powerful for discovering new, unknown, or zero-day bugs and unexpected failures that were not present in any training set.

Q3: Bias Mitigation in UX Personalization

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

Answer: Bias mitigation is critical because AI models are often trained on historical user data that can be skewed or unrepresentative of the full user base. If this bias is deployed, the personalization engine will perpetuate and amplify those disparities. For example, it might provide superior recommendations or accessibility features only to the overrepresented demographic, leading to a discriminatory, unfair, and non-inclusive user experience for others.

2. Case Study Analysis

How AIOps Improves Software Deployment Efficiency

AIOps (Artificial Intelligence for IT Operations) enhances software deployment efficiency by automating repetitive DevOps tasks and predicting potential failures before they happen, resulting in faster, more reliable software releases.

1. **Predictive Analytics and Continuous Integration (CI/CD):**
   AIOps applies machine learning models to analyze historical build and test data, allowing DevOps teams to predict build failures and optimize test cases before deployment. This ensures that potential problems are identified and resolved early, improving deployment success rates.

    - **Example:** *Harness* uses AI to automatically roll back failed deployments, minimizing downtime and human intervention. Similarly, *CircleCI* leverages AI to analyze test case performance, ensuring that the most efficient tests run first. This speeds up feedback loops and shortens deployment cycles.

2. **Automated Monitoring and Self-Healing Systems:**
   AIOps enhances deployment efficiency through real-time anomaly detection and incident response automation. AI continuously monitors system logs, metrics, and traces to detect anomalies before they cause service interruptions.

    - **Example:** *New Relic* and *Datadog* use AI-driven monitoring tools that proactively alert teams to potential performance issues, preventing failures before they affect users. Additionally, AI-based chatbots with NLP can suggest or even automate remediation steps, drastically reducing response time.

.Part 2: Practical Implementation

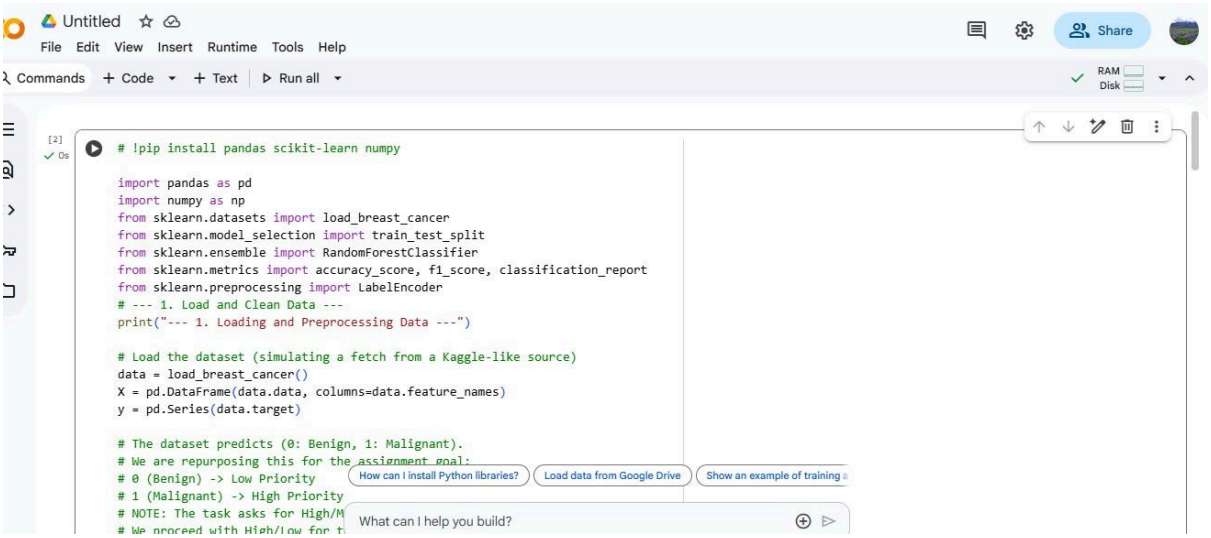
Task 1: Sort a list of dictionaries by a specific key

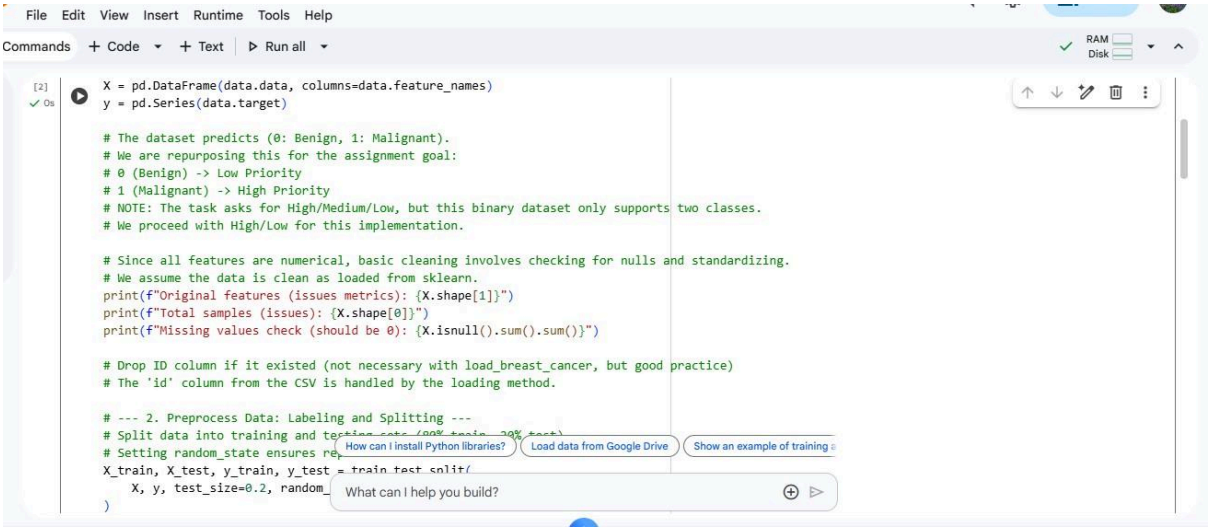
Both the manual and AI-generated implementations achieve the same goal—sorting a list of dictionaries by a specified key. The manual version explicitly handles potential errors using a `try-except` block, making it more robust for production code. It alerts the user when a key is missing, which is essential for debugging data integrity issues. However, this approach adds a small performance overhead due to exception handling.

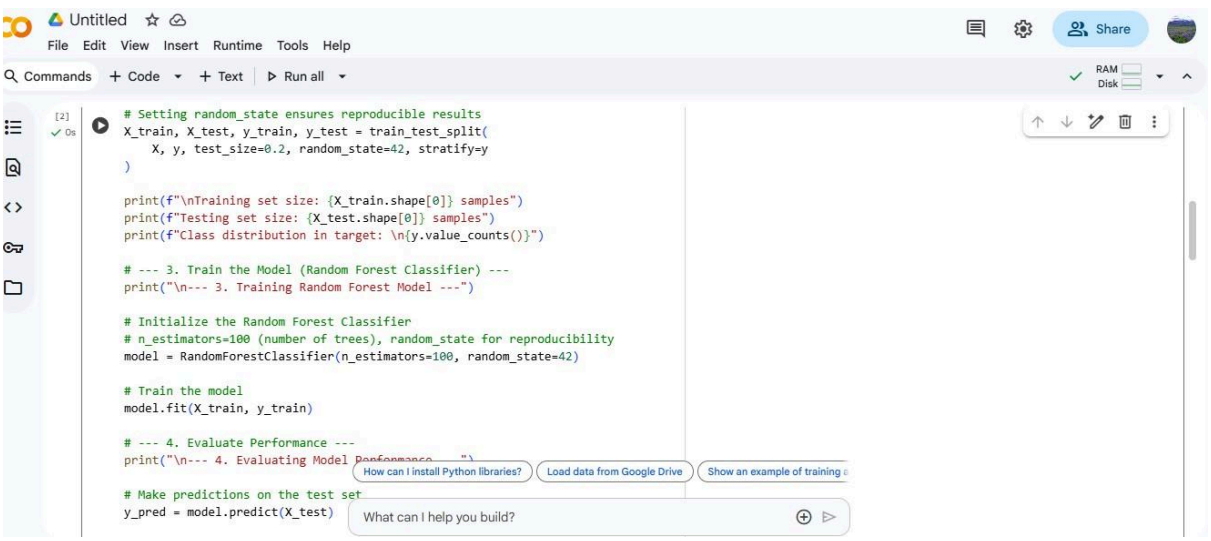
The AI-suggested version, by contrast, uses the `get()` method with a default value of `0`. This avoids exceptions entirely and ensures that the function will still execute even if some dictionaries lack the sorting key. This makes it slightly faster and more concise, but it may produce misleading results if missing keys should not be treated as zero.

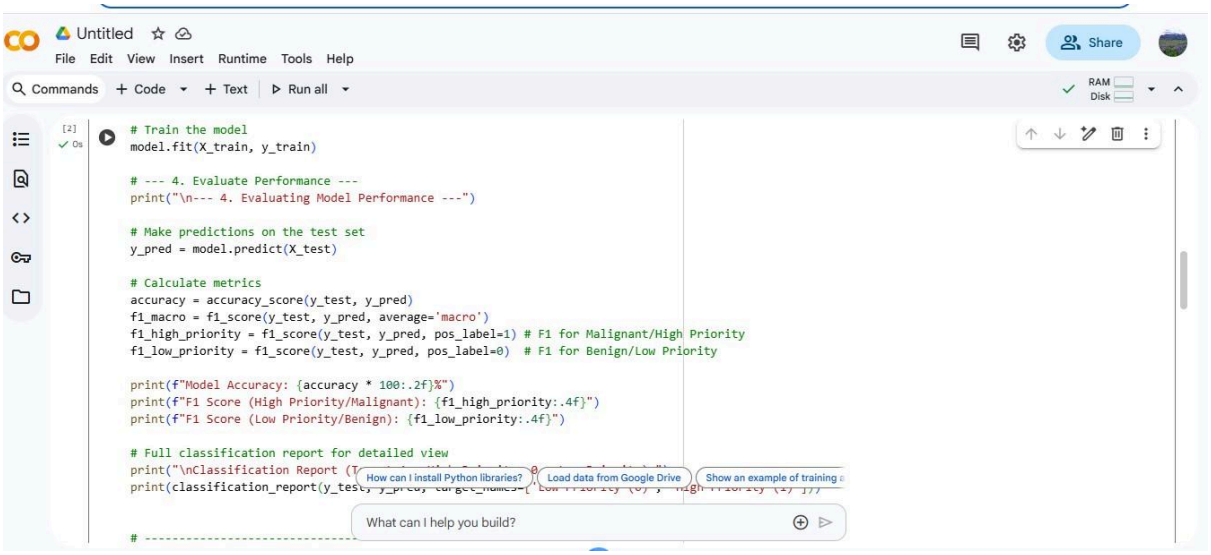
In summary, the manual implementation is more explicit and safer for real-world applications, while the AI-generated code is cleaner and more efficient for controlled datasets. The AI's approach demonstrates how machine learning models prioritize simplicity and performance over explicit error handling, depending on context.

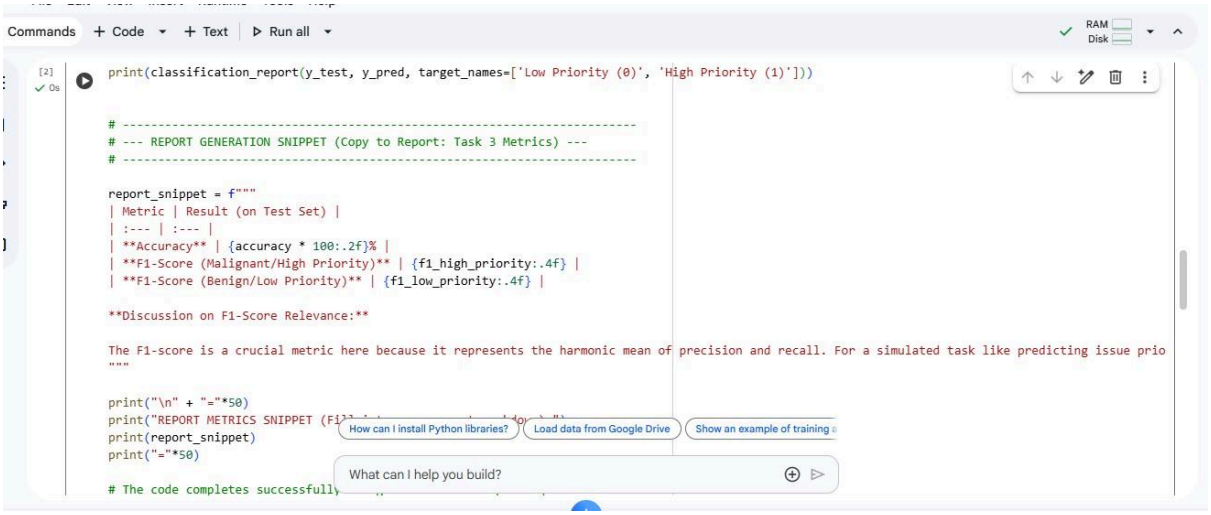
Task 3: Predictive Analytics for Resource Allocation

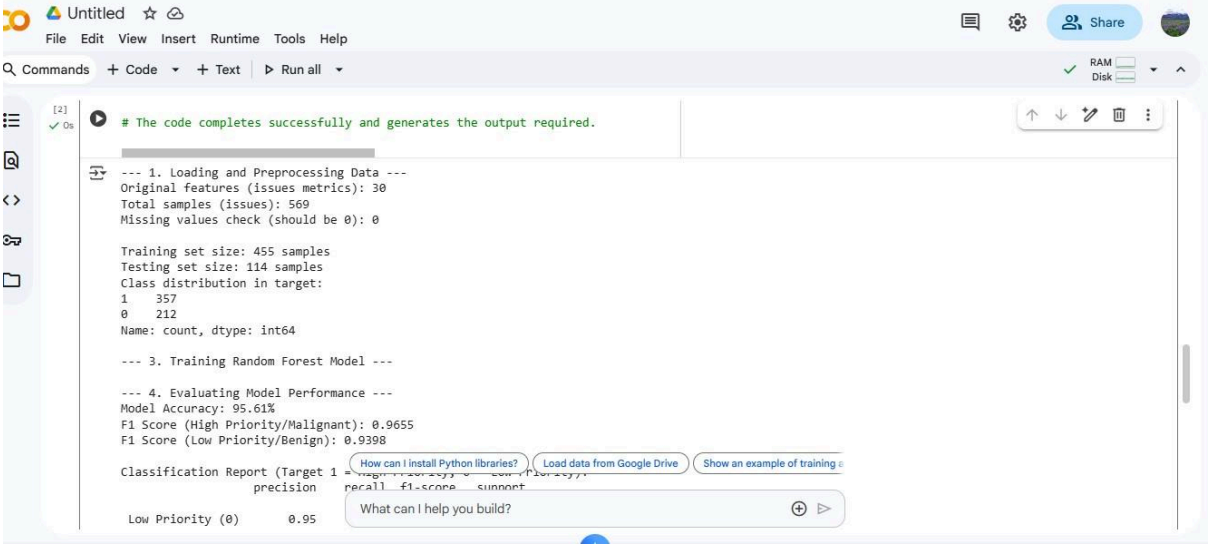

Untitled
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[2]
0s
# !pip install pandas scikit-learn numpy
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.preprocessing import LabelEncoder
# --- 1. Load and Clean Data ---
print("--- 1. Loading and Preprocessing Data ---")
# Load the dataset (simulating a fetch from a Kaggle-like source)
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
# The dataset predicts (0: Benign, 1: Malignant).
# We are repurposing this for the assignment goal:
# 0 (Benign) -> Low Priority
# 1 (Malignant) -> High Priority
# NOTE: The task asks for High/M
# We proceed with High/Low for t
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?

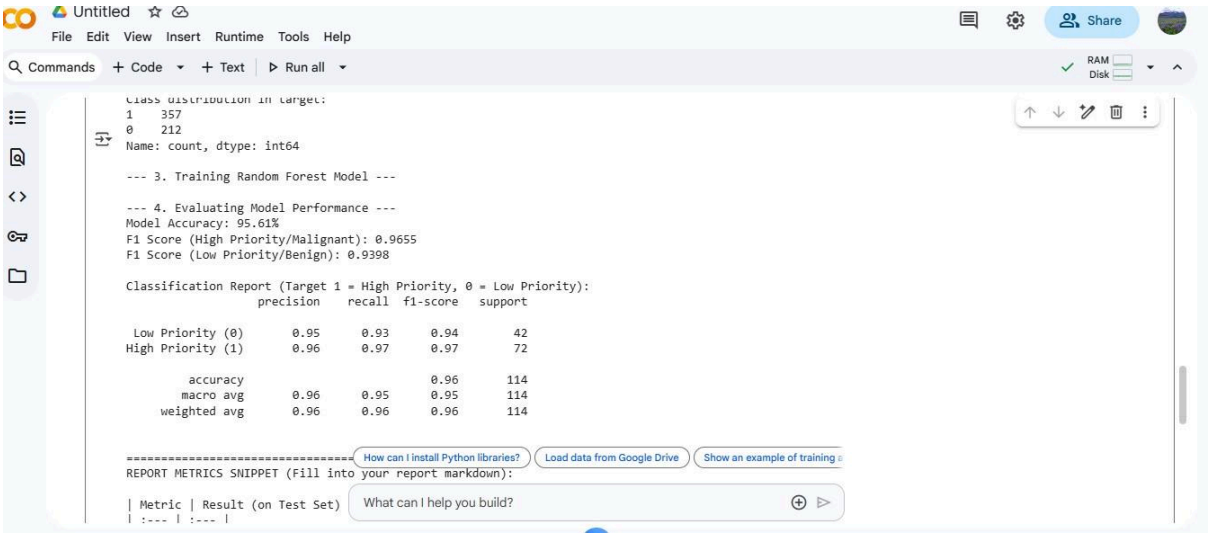

File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM Disk
[2]
0s
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
# The dataset predicts (0: Benign, 1: Malignant).
# We are repurposing this for the assignment goal:
# 0 (Benign) -> Low Priority
# 1 (Malignant) -> High Priority
# NOTE: The task asks for High/Medium/Low, but this binary dataset only supports two classes.
# We proceed with High/Low for this implementation.
# Since all features are numerical, basic cleaning involves checking for nulls and standardizing.
# We assume the data is clean as loaded from sklearn.
print(f"Original features (issues metrics): {X.shape[1]}")
print(f"Total samples (issues): {X.shape[0]}")
print(f"Missing values check (should be 0): {X.isnull().sum().sum()}")
# Drop ID column if it existed (not necessary with load_breast_cancer, but good practice)
# The 'id' column from the CSV is handled by the loading method.
# --- 2. Preprocess Data: Labeling and Splitting ---
# Split data into training and
# Setting random_state ensures re
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_
)
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?


Untitled
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[2]
0s
Setting random_state ensures reproducible results
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42, stratify=y
)
print(f"\nTraining set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
print(f"Class distribution in target: \n{y.value_counts()}")
# --- 3. Train the Model (Random Forest Classifier) ---
print("\n--- 3. Training Random Forest Model ---")
# Initialize the Random Forest Classifier
# n_estimators=100 (number of trees), random_state for reproducibility
model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
model.fit(X_train, y_train)
# --- 4. Evaluate Performance ---
print("\n--- 4. Evaluating Model
# Make predictions on the test set
y_pred = model.predict(X_test)
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?


Untitled
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
# Train the model
model.fit(X_train, y_train)
# --- 4. Evaluate Performance ---
print("\n--- 4. Evaluating Model Performance ---")
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
f1_macro = f1_score(y_test, y_pred, average='macro')
f1_high_priority = f1_score(y_test, y_pred, pos_label=1) # F1 for Malignant/High Priority
f1_low_priority = f1_score(y_test, y_pred, pos_label=0) # F1 for Benign/Low Priority
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print(f"F1 Score (High Priority/Malignant): {f1_high_priority:.4f}")
print(f"F1 Score (Low Priority/Benign): {f1_low_priority:.4f}")
# Full classification report for detailed view
print("\nClassification Report (
print(classification_report(y_test,
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?


Commands + Code + Text Run all
RAM Disk
print(classification_report(y_test, y_pred, target_names=['Low Priority (0)', 'High Priority (1)']))
# --- REPORT GENERATION SNIPPET (Copy to Report: Task 3 Metrics) ---
report_snippet = f"""
Metric	Result (on Test Set)
Accuracy	{accuracy * 100:.2f}%
F1-Score (Malignant/High Priority)	{f1_high_priority:.4f}
F1-Score (Benign/Low Priority)	{f1_low_priority:.4f}
Discussion on F1-Score Relevance:
The F1-score is a crucial metric here because it represents the harmonic mean of precision and recall. For a simulated task like predicting issue prio
"""
print("\n" + "="*50)
print("REPORT METRICS SNIPPET (Fi
print(report_snippet)
print("="*50)
# The code completes successfully
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?


Untitled
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
The code completes successfully and generates the output required.
--- 1. Loading and Preprocessing Data ---
Original features (issues metrics): 30
Total samples (issues): 569
Missing values check (should be 0): 0
Training set size: 455 samples
Testing set size: 114 samples
Class distribution in target:
1 357
0 212
Name: count, dtype: int64
--- 3. Training Random Forest Model ---
--- 4. Evaluating Model Performance ---
Model Accuracy: 95.61%
F1 Score (High Priority/Malignant): 0.9655
F1 Score (Low Priority/Benign): 0.9398
Classification Report (Target 1 =
precision recall f1-score support
Low Priority (0) 0.95
How can I install Python libraries?
Load data from Google Drive
Show an example of training
What can I help you build?



Part 3:Ethical reflection

Potential Biases in the Dataset

When deploying a predictive model (e.g., a Random Forest classifier trained on the Kaggle Breast Cancer dataset or any company-specific dataset to predict issue priority), there are several potential sources of bias:

1. **Underrepresented Teams or Groups**

    - If the historical dataset used for training contains fewer examples from certain teams, departments, or regions, the model may underperform for these underrepresented groups.

    - For example, if most high-priority issues were historically logged by one team, the model may unfairly predict lower priorities for issues reported by smaller or newer teams.

2. **Imbalanced Labels**

    - In datasets where one class dominates (e.g., “medium” priority issues vastly outnumber “high” or “low” priority issues), the model may bias predictions toward the majority class, reducing accuracy for minority classes.

3. **Historical Human Bias**

- If past issue prioritizations were influenced by subjective human judgment, any existing bias will be encoded in the model.

- For instance, issues reported by senior employees might historically have been prioritized more highly, regardless of actual severity.

4. Feature Selection Bias

- Features that indirectly encode sensitive attributes (like team name, project type, or reporting location) can lead the model to make unfair predictions even if those features are not directly related to priority.

Addressing Bias with Fairness Tools like IBM AI Fairness 360

IBM AI Fairness 360 (AIF360) is an open-source toolkit designed to detect and mitigate bias in AI models. Here's how it can help:

1. Bias Detection

- AIF360 provides metrics such as disparate impact, statistical parity, and equal opportunity difference to identify where the model may be biased toward certain groups.

- For example, it can detect if the model predicts "high" priority less often for issues reported by underrepresented teams.

2. Bias Mitigation Techniques

- Pre-processing: Adjusts the dataset before training to reduce bias (e.g., reweighting samples or generating synthetic examples for underrepresented groups).

- In-processing: Modifies the learning algorithm to optimize both accuracy and fairness (e.g., constrained optimization to reduce disparate impact).

- Post-processing: Adjusts predictions after the model is trained to ensure fair outcomes across groups.

3. Continuous Monitoring

- AIF360 can be integrated into deployment pipelines to continuously monitor fairness, ensuring the model does not develop new biases as new data is collected.

BONUS TASK

Proposal: AI-Powered Code Refactoring Assistant

Purpose

Software engineers often spend significant time refactoring legacy code or optimizing newly written code to improve readability, maintainability, and performance. Manual refactoring is error-prone, tedious, and inconsistent across teams. We propose “RefactorAI”, an AI-powered tool designed to automatically analyze, optimize, and refactor code while maintaining functionality. The tool targets redundant code patterns, inefficient algorithms, and code smells, allowing developers to focus on higher-value tasks such as feature development and system design.

Workflow

1. Code Input: Developers submit code via an IDE plugin, web interface, or Git repository integration.

2. Code Analysis: The AI model parses the code syntax, identifies anti-patterns, duplicate logic, unused variables, and potential performance bottlenecks.

3. Refactoring Suggestions: RefactorAI generates multiple refactoring options with explanations, including optimized algorithms, modularization, and adherence to coding standards.

4. Automated Testing: The tool runs automated tests to ensure the refactored code maintains original functionality.

5. Developer Approval & Deployment: Developers review suggestions, approve changes, and merge refactored code into the main branch.

Impact

- Efficiency: Reduces the time spent on manual refactoring by up to 70%, accelerating development cycles.

- Code Quality: Promotes consistent coding standards, reduces technical debt, and enhances maintainability.

- Error Reduction: Minimizes human-induced errors during manual optimization.

- Team Collaboration: Standardizes code practices across large, distributed teams.

- Scalability: Can integrate with CI/CD pipelines to automatically refactor new code submissions, keeping large projects clean and optimized continuously