QUIZ 1:

How do AI-driven code generation tools reduce development time? What are their limitations?

AI-driven code generation tools **reduce development time** by:

- **Auto-completing code snippets** based on context, reducing manual typing.
- **Suggesting best practices** (e.g., error handling, optimizations).
- **Accelerating boilerplate code** (e.g., API routes, class definitions).

Limitations:

- **Security risks** .may suggest vulnerable code if trained on flawed examples.
- **Lack of deep understanding** .can produce syntactically correct but logically wrong code.
- **Over-reliance risk** .junior developers may not learn fundamentals.

QUIZ 2:

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

Supervised learning for automated bug detection

Supervised learning for bug detection relies on **labeled data**. This means you need a dataset where each piece of code (or bug report, or system log) is explicitly tagged as either "buggy" or "non-buggy," and often, if buggy, the *type* of bug. The algorithm learns to map input features (e.g., code metrics, static analysis warnings, commit messages) to these predefined labels.

How it Works:

1. **Data Collection & Labeling:** Gather a large dataset of historical code, test cases, bug reports, and their resolutions. Crucially, human experts (developers, testers) must manually label each instance as "buggy" or "clean." For example, a code snippet would be labeled "buggy" if it led to a known defect, and "clean" otherwise.
2. **Feature Engineering:** Extract relevant features from the code or data. These could include lines of code, cyclomatic complexity, number of changes, author, historical bug density in a module, use of specific API calls, or even semantic features derived from code analysis.
3. **Model Training:** A machine learning model (e.g., Decision Trees, Random Forests, Support Vector Machines, Neural Networks) is trained on this labeled dataset. The model learns patterns and correlations between the features and the "buggy" or "clean" labels.
4. **Prediction:** Once trained, the model can be used to predict whether new, unseen code is likely to contain bugs.

Unsupervised learning for automated bug detection

Unsupervised learning for bug detection works with **unlabeled data**. Instead of learning from examples of "buggy" and "clean" code, it aims to discover hidden patterns, structures, or anomalies within the code or system behavior itself. Deviations from these learned "normal" patterns are then flagged as potential bugs.

How it Works:

1. **Data Collection:** Gather vast amounts of unlabeled code, execution traces, log files, or version control history. No manual labeling is required upfront.
2. **Feature Extraction:** Similar to supervised learning, features are extracted. These might focus on code structure, data flow, control flow, or statistical properties of logs/metrics.
3. **Pattern Discovery:** An unsupervised learning algorithm (e.g., Clustering, Anomaly Detection, Dimensionality Reduction) analyzes the data to find inherent groupings, correlations, or outliers.
    - **Clustering:** Groups similar pieces of code together. Code that doesn't fit well into any cluster, or forms a tiny, isolated cluster, might be anomalous.
    - **Anomaly Detection:** Learns what "normal" code or behavior looks like and flags anything that significantly deviates from this norm.

    - **Dimensionality Reduction:** Simplifies complex code data into lower-dimensional representations, making it easier to visualize and identify unusual patterns.
4. **Anomaly Flagging:** The algorithm identifies data points that are outliers or do not conform to the established patterns. These anomalies are then presented to developers for further investigation as potential bugs.

In simper explanation :

- **Supervised:** Train a model on historical bug reports to predict new ones.
- **Unsupervised:** Cluster code commits to find unusual patterns.

Quiz 3:

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

- **Discriminatory recommendations** e.g., favoring one demographic over another.
- **Reinforcing stereotypes** e.g. gender-biased job ads.
- **Legal & reputational risks** e.g. violating fairness regulations like GDPR.