Multithreaded Dataset Insertion in Python

Imports and Global Variables

```
import threading
import random
import time
from collections import defaultdict
```

This section imports the necessary modules:
- `threading` for handling concurrent operations.
- `random` and `time` for simulating random entries and delay.
- `defaultdict` to count distribution frequencies later.

Shared Resources and Lock

```
# Shared dataset where all threads will add entries
dataset = []

# Binary lock to ensure only one thread writes to the dataset at a time
lock = threading.Lock()
```

`dataset` is a shared list where threads add data. `lock` ensures that only one thread writes at a time, preventing race conditions.

Field Initialization

```
# Valid values for fields
years = [2022, 2023, 2024]
courses = set()
universities = set()
```

The possible values for 'Year' are pre-defined. `courses` and `universities` are empty sets to be filled with unique values from the input file.

Load Unique Fields from File

```
# Step 1: Load unique Course Names and University names from file
def load_unique_fields(filename):

    with open(filename, 'r') as file:
        for line in file:
            parts = [p.strip() for p in line.strip().split(',')]
            if len(parts) == 4:
                _, course, _, university = parts
                courses.add(course)
                universities.add(university)
```

Reads a file and extracts unique course and university names assuming each line is in the format:
Year, Course, Grade, University.

Thread Function to Add Entries

```
# Step 2: Thread function to add entries safely
def add_entries(thread_id, entries_per_thread):

    for _ in range(entries_per_thread):
        entry = {
            "Year": random.choice(years),
            "Course Name": random.choice(list(courses)),
            "Grade": random.randint(60, 100),
            "University": random.choice(list(universities))
        }


        with lock:
            dataset.append(entry)

        time.sleep(0.01)
```

This function is run by each thread to add entries to the shared dataset.
The lock is used to prevent concurrent write issues.

Simulate Concurrent Addition

```python
# Step 3: Simulate multiple threads concurrently adding entries
def simulate_concurrent_addition(num_threads=10, entries_per_thread=5):

    threads = []
    for i in range(num_threads):
        t = threading.Thread(target=add_entries, args=(i, entries_per_thread))
        threads.append(t)
        t.start()

    # Wait for all threads to finish
    for t in threads:
        t.join()
```

Creates and starts multiple threads that run concurrently, then waits for all to complete.

Verify Dataset Consistency

```python
# Step 4: Verify consistency and distribution of the final dataset
def verify_dataset(num_threads, entries_per_thread):
    expected = num_threads * entries_per_thread
    print(" Dataset Verification ")
    print(f"Expected entries: {expected}")
    print(f"Actual entries:   {len(dataset)}")

    # Count distribution
    course_dist = defaultdict(int)
    year_dist = defaultdict(int)
    university_dist = defaultdict(int)

    for entry in dataset:
        course_dist[entry["Course Name"]] += 1
        year_dist[entry["Year"]] += 1
        university_dist[entry["University"]] += 1

    print("\n Entries per Course:")
    for course, count in course_dist.items():
        print(f"  {course}: {count}")

    print("\n Entries per Year:")
    for year, count in sorted(year_dist.items()):
        print(f"   {year}: {count}")

    print("\n Entries per University:")
    for uni, count in sorted(university_dist.items()):
        print(f"  {uni}: {count}")
```

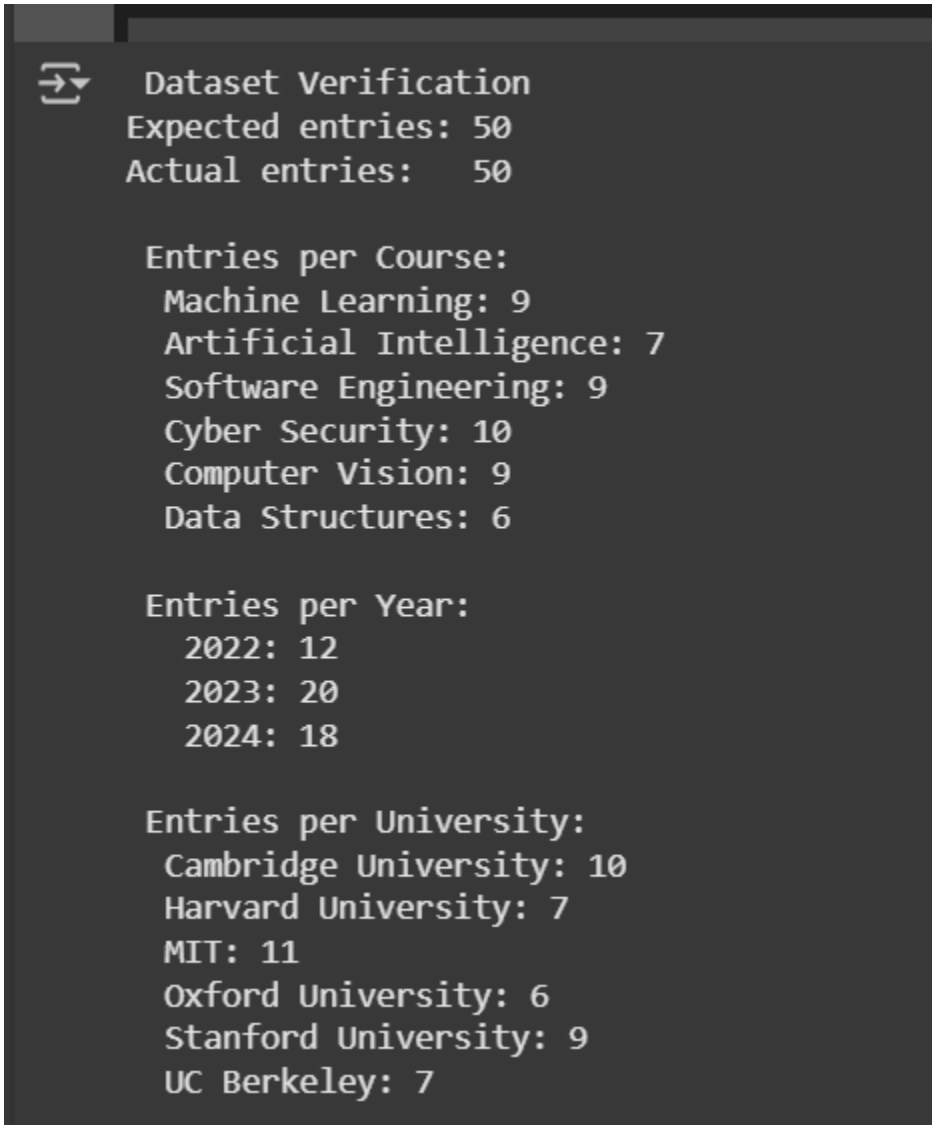
Checks if the number of inserted entries matches the expected count.
It also counts how many entries exist for each course, year, and university.

Main Execution Block

```
# Main execution
if __name__ == "__main__":

    load_unique_fields("coursegrades.txt")

    courses = list(courses)
    universities = list(universities)

    simulate_concurrent_addition(num_threads=10, entries_per_thread=5)

    verify_dataset(num_threads=10, entries_per_thread=5)
```

Runs the full process: loads input, simulates threaded data insertion, and verifies the result and takes as an input data from coursegrades.txt.

Two consecutive outputs with lock



```
 Dataset Verification
Expected entries: 50
Actual entries:   50

 Entries per Course:
  Cyber Security: 11
  Computer Vision: 8
  Artificial Intelligence: 9
  Data Structures: 14
  Machine Learning: 5
  Software Engineering: 3

 Entries per Year:
   2022: 21
   2023: 10
   2024: 19

 Entries per University:
  Cambridge University: 5
  Harvard University: 5
  MIT: 9
  Oxford University: 11
  Stanford University: 12
  UC Berkeley: 8
```

Two consecutive outputs without lock

```
 Dataset Verification
Expected entries: 50
Actual entries:   50

 Entries per Course:
  Artificial Intelligence: 7
  Machine Learning: 15
  Computer Vision: 7
  Software Engineering: 11
  Data Structures: 4
  Cyber Security: 6

 Entries per Year:
   2022: 15
   2023: 20
   2024: 15

 Entries per University:
  Cambridge University: 9
  Harvard University: 7
  MIT: 9
  Oxford University: 6
  Stanford University: 7
  UC Berkeley: 12
```

```
 Dataset Verification
Expected entries: 50
Actual entries:   50

 Entries per Course:
  Machine Learning: 12
  Artificial Intelligence: 10
  Data Structures: 7
  Computer Vision: 5
  Cyber Security: 8
  Software Engineering: 8

 Entries per Year:
   2022: 16
   2023: 20
   2024: 14

 Entries per University:
  Cambridge University: 10
  Harvard University: 6
  MIT: 8
  Oxford University: 6
  Stanford University: 9
  UC Berkeley: 11
```