

# AI – ASSISTED CODING

## LAB EXAM - 2

HALLTICKET NO: 2403A51410

NAME: A.VIKAS NAYAK

A.1 — [S15A1] Compute per-viewer average from logs (AI completion)

Scenario (digital media streaming):

Context:

You are integrating a digital media streaming telemetry service where each viewer emits

periodic measures as CSV lines: `id,timestamp,watch\_time`. Due to flaky connectivity, some

lines may be truncated or contain non-numeric values. Ops needs a quick aggregation for

dashboards and alert thresholds

# AI – ASSISTED CODING

## LAB EXAM - 2

### PROMPT:

Write a Python function to parse the raw text (multiple lines) and compute per-viewer averages of `watch\_time`. Return a dict {id: avg} and separately compute an overall average.

I'll create a Python function to parse the raw CSV telemetry data and compute per-viewer averages with robust error handling for flaky connectivity issues.

Tool edit\_file not found.

I'll use the write tool to create the comprehensive streaming telemetry parser:

LABEXAM-2-1.py +173 -1 •

Now let me test the implementation to make sure it works correctly:

Run command: cd, python

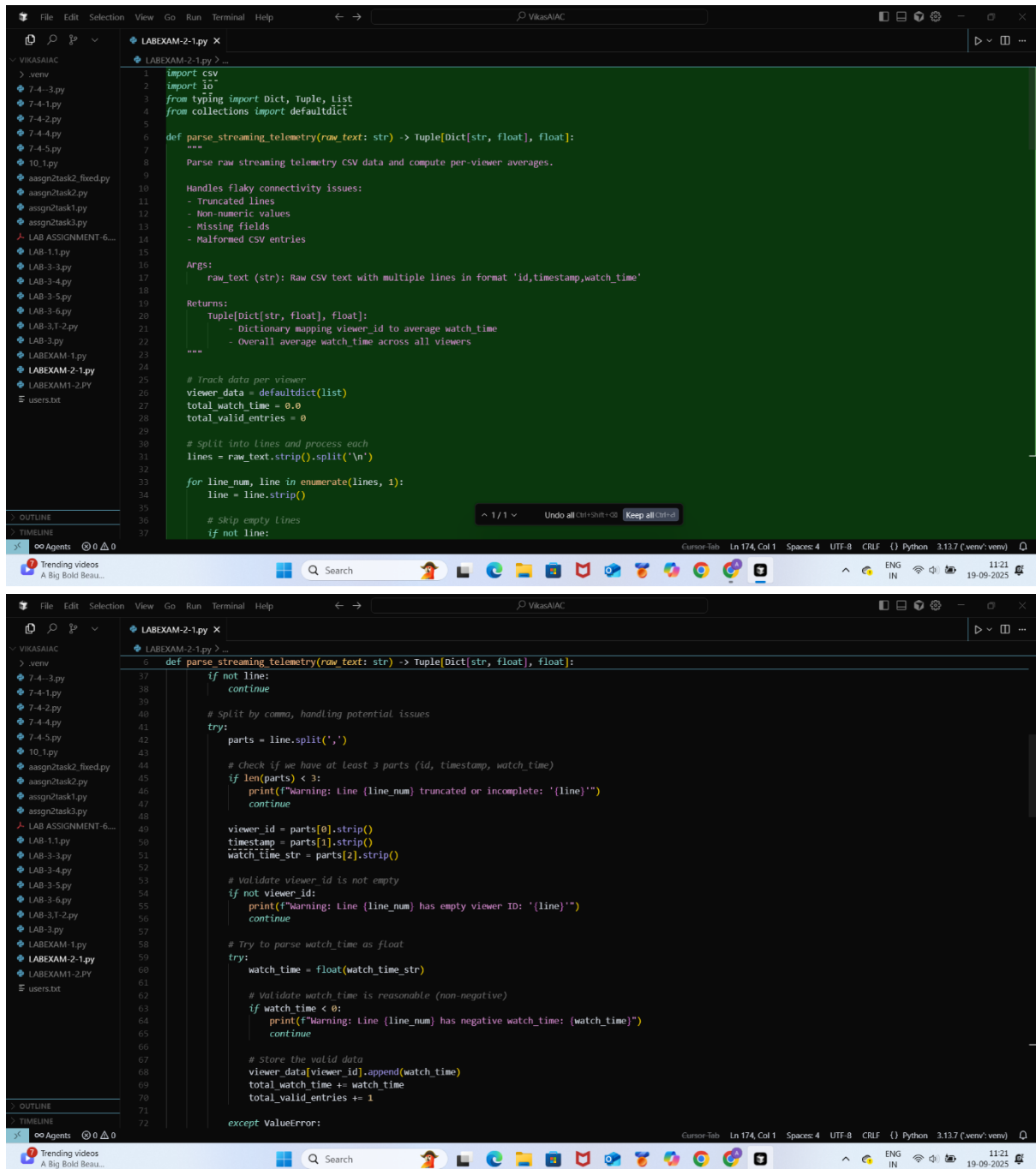
```
$ cd
"C:\Users\vikas\OneDrive\Desktop\VikasAIAC"
" && python LABEXAM-2-1.py
```

Skip Add 'cd' + 1 to Allowlist Run

# AI – ASSISTED CODING

## LAB EXAM - 2

### Ai-generated code:



The image displays two screenshots of a code editor (Cursor) showing AI-generated Python code for parsing streaming telemetry data. The code is written in a dark-themed editor with a file explorer on the left and a terminal at the bottom.

**Top Screenshot:** Shows the initial code structure. The file explorer on the left lists files like `LABEXAM-2-1.py`, `LABEXAM-1.py`, and `LABEXAM-2-PY`. The code in the editor defines a function `parse_streaming_telemetry` that takes `raw_text` as input and returns a tuple containing a dictionary of viewer data, the total watch time, and the total valid entries. The code includes comments about handling flaky connectivity issues and splitting the input into lines.

```
1 import csv
2 import io
3 from typing import Dict, Tuple, List
4 from collections import defaultdict
5
6 def parse_streaming_telemetry(raw_text: str) -> Tuple[Dict[str, float], float]:
7     """
8     Parse raw streaming telemetry CSV data and compute per-viewer averages.
9
10    Handles flaky connectivity issues:
11    - Truncated lines
12    - Non-numeric values
13    - Missing fields
14    - Malformed CSV entries
15
16    Args:
17        raw_text (str): Raw CSV text with multiple lines in format 'id,timestamp,watch_time'
18
19    Returns:
20        Tuple[Dict[str, float], float]:
21            - Dictionary mapping viewer id to average watch time
22            - Overall average watch time across all viewers
23    """
24
25    # Track data per viewer
26    viewer_data = defaultdict(list)
27    total_watch_time = 0.0
28    total_valid_entries = 0
29
30    # Split into lines and process each
31    lines = raw_text.strip().split('\n')
32
33    for line_num, line in enumerate(lines, 1):
34        line = line.strip()
35
36        # Skip empty lines
37        if not line:
```

**Bottom Screenshot:** Shows the continuation of the `parse_streaming_telemetry` function. The code handles potential issues like truncated lines, missing fields, and malformed CSV entries. It splits the input by comma, checks if there are at least 3 parts (id, timestamp, watch\_time), and validates the viewer ID and watch time. The code then stores the valid data and updates the total watch time and total valid entries.

```
38        if not line:
39            continue
40
41        # Split by comma, handling potential issues
42        try:
43            parts = line.split(',')
44
45            # Check if we have at least 3 parts (id, timestamp, watch_time)
46            if len(parts) < 3:
47                print(f"Warning: Line (line_num) truncated or incomplete: '{line}'")
48                continue
49
50            viewer_id = parts[0].strip()
51            timestamp = parts[1].strip()
52            watch_time_str = parts[2].strip()
53
54            # Validate viewer_id is not empty
55            if not viewer_id:
56                print(f"Warning: Line (line_num) has empty viewer ID: '{line}'")
57                continue
58
59            # Try to parse watch_time as float
60            try:
61                watch_time = float(watch_time_str)
62
63                # Validate watch_time is reasonable (non-negative)
64                if watch_time < 0:
65                    print(f"Warning: Line (line_num) has negative watch_time: {watch_time}")
66                    continue
67
68                # Store the valid data
69                viewer_data[viewer_id].append(watch_time)
70                total_watch_time += watch_time
71                total_valid_entries += 1
72
73    except ValueError:
```

```
File Edit Selection View Go Run Terminal Help
LABEXAM-2-1.py x
LABEXAM-2-1.py -> ...
5 def parse_streaming_telemetry(row_text: str) -> Tuple[Dict[str, float], float]:
6
7     except ValueError:
8         print(f"Warning: Line {line_num} has non-numeric watch_time '{watch_time_str}': '{line}'")
9         continue
10
11     except Exception as e:
12         print(f"Warning: Line {line_num} caused parsing error: {e} - '{line}'")
13         continue
14
15     # Compute per-viewer averages
16     viewer_averages = {}
17     for viewer_id, watch_times in viewer_data.items():
18         if watch_times: # Safety check
19             viewer_averages[viewer_id] = sum(watch_times) / len(watch_times)
20
21     # Compute overall average
22     overall_average = total_watch_time / total_valid_entries if total_valid_entries > 0 else 0.0
23
24     return viewer_averages, overall_average
25
26 def generate_dashboard_summary(viewer_averages: Dict[str, float], overall_average: float) -> Dict:
27     """
28     Generate dashboard-ready summary statistics.
29
30     Args:
31         viewer_averages: Dictionary of viewer_id to average watch_time
32         overall_average: Overall average watch_time
33
34     Returns:
35         Dictionary with summary statistics for dashboards and alerts
36     """
37
38     if not viewer_averages:
39         return {
40             "total viewers": 0,
41             "overall average": 0.0,
42         }
```

```
File Edit Selection View Go Run Terminal Help
LABEXAM-2-1.py x
LABEXAM-2-1.py ~
92 def generate_dashboard_summary(viewer_averages: Dict[str, float], overall_average: float) -> Dict:
106     "total_viewers": 0,
107     "overall_average": 0.0,
108     "min_average": 0.0,
109     "max_average": 0.0,
110     "alert_thresholds": {
111         "low_engagement": 0,
112         "high_engagement": 0
113     }
114
115     watch_times = list(viewer_averages.values())
116
117     summary = {
118         "total_viewers": len(viewer_averages),
119         "overall_average": round(overall_average, 2),
120         "min_average": round(min(watch_times), 2),
121         "max_average": round(max(watch_times), 2),
122         "median_average": round(sorted(watch_times)[len(watch_times)//2], 2),
123         "alert_thresholds": {
124             "low_engagement": len([wt for wt in watch_times if wt < overall_average * 0.5]),
125             "high_engagement": len([wt for wt in watch_times if wt > overall_average * 1.5])
126         }
127     }
128
129     return summary
130
131 # Example usage and test cases
132
133 if __name__ == "__main__":
134     # Test with sample data including various error conditions
135     sample_data = """user1,2024-01-01T10:00:00,120.5
136 user2,2024-01-01T10:01:00,65.2
137 user1,2024-01-01T10:02:00,95.0
138 user3,2024-01-01T10:03:00,200.1
139 user2,2024-01-01T10:04:00,invalid_time
140 user4,2024-01-01T10:05:00,150.8
141 """
```

# AI – ASSISTED CODING

## LAB EXAM - 2

```
140 user2,2024-01-01T10:04:00,invalid_time
141 user4,2024-01-01T10:05:00,150.0
142 user1,2024-01-01T10:06:00,110.3
143 truncated_line,2024-01-01T10:07:00
144 user5,2024-01-01T10:08:00,-50.0
145 user6,2024-01-01T10:09:00,75.8
146 user2,2024-01-01T10:10:00,90.5
147 user7,2024-01-01T10:11:00,180.2
148 user8,2024-01-01T10:12:00,abc
149 user9,2024-01-01T10:13:00,160.7
150 user10,2024-01-01T10:14:00,140.0"""
151
152 print("=== Streaming Telemetry Parser Test ===\n")
153
154 # Parse the data
155 viewer_averages, overall_average = parse_streaming_telemetry(sample_data)
156
157 print("Per-viewer averages:")
158 for viewer_id, avg_time in sorted(viewer_averages.items()):
159     print(f" {viewer_id}: {avg_time:.2f} seconds")
160
161 print(f"\nOverall average: {overall_average:.2f} seconds")
162
163 # Generate dashboard summary
164 summary = generate_dashboard_summary(viewer_averages, overall_average)
165
166 print("\n=== Dashboard Summary ===")
167 print(f"Total viewers: {summary['total_viewers']}")
168 print(f"Overall average: {summary['overall_average']} seconds")
169 print(f"Min average: {summary['min_average']} seconds")
170 print(f"Max average: {summary['max_average']} seconds")
171 print(f"Median average: {summary['median_average']} seconds")
172 print(f"Low engagement viewers (< 50% of avg): {summary['alert_thresholds']['low_engagement']}")
173 print(f"High engagement viewers (> 150% of avg): {summary['alert_thresholds']['high_engagement']}")
174
```

Output:

```
PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC> & c:\Users\vikas\OneDrive\Desktop\VikasAIAC\.venv\Scripts\Activate.ps1
(.venv) PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC> & c:\Users\vikas\OneDrive\Desktop\VikasAIAC\.venv\Scripts\python.exe c:\Users\vikas\OneDrive\Desktop\VikasAIAC\LAB
EXAM-2-1.py
=== Streaming Telemetry Parser Test ===

Warning: Line 5 has non-numeric watch time 'invalid_time': 'user2,2024-01-01T10:04:00,invalid_time'
Warning: Line 8 truncated or incomplete: 'truncated_line,2024-01-01T10:07:00'
Warning: Line 9 has negative watch time: '-50.0'
Warning: Line 13 has non-numeric watch time 'abc': 'user8,2024-01-01T10:12:00,abc'
Per-viewer averages:
user1: 180.60 seconds
user10: 140.00 seconds
user2: 87.85 seconds
user3: 200.10 seconds
Total viewers: 8
Overall average: 129.03 seconds
Min average: 75.0 seconds
Max average: 200.1 seconds
Median average: 150.0 seconds
Low engagement viewers (< 50% of avg): 0
High engagement viewers (> 150% of avg): 1
(.venv) PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC>
```

# AI – ASSISTED CODING

## LAB EXAM - 2

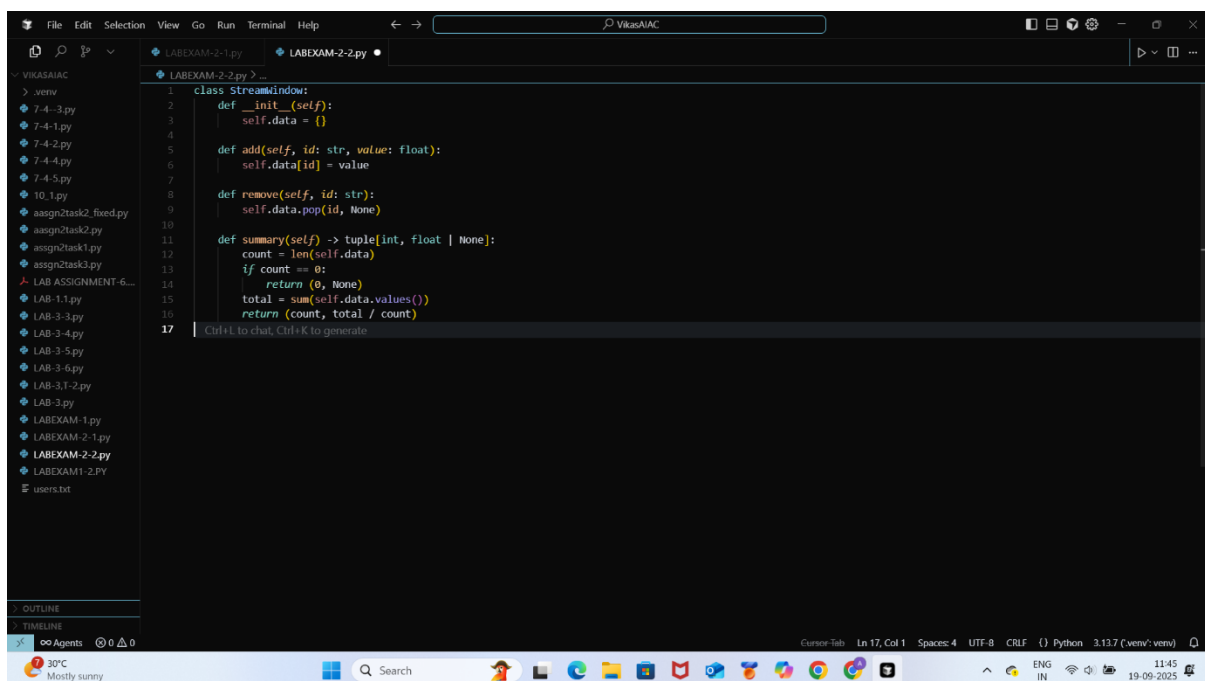
### A.2 — [S15A2] Implement StreamWindow with add/remove/summary (AI completion)

Scenario (digital media streaming):

Context:

A microservice in the digital media streaming platform maintains a small in-memory structure to track values keyed by identifier (e.g., order IDs, sensor IDs). Engineers want a minimal class to add, remove, and summarize current values for quick health checks.

CODE:



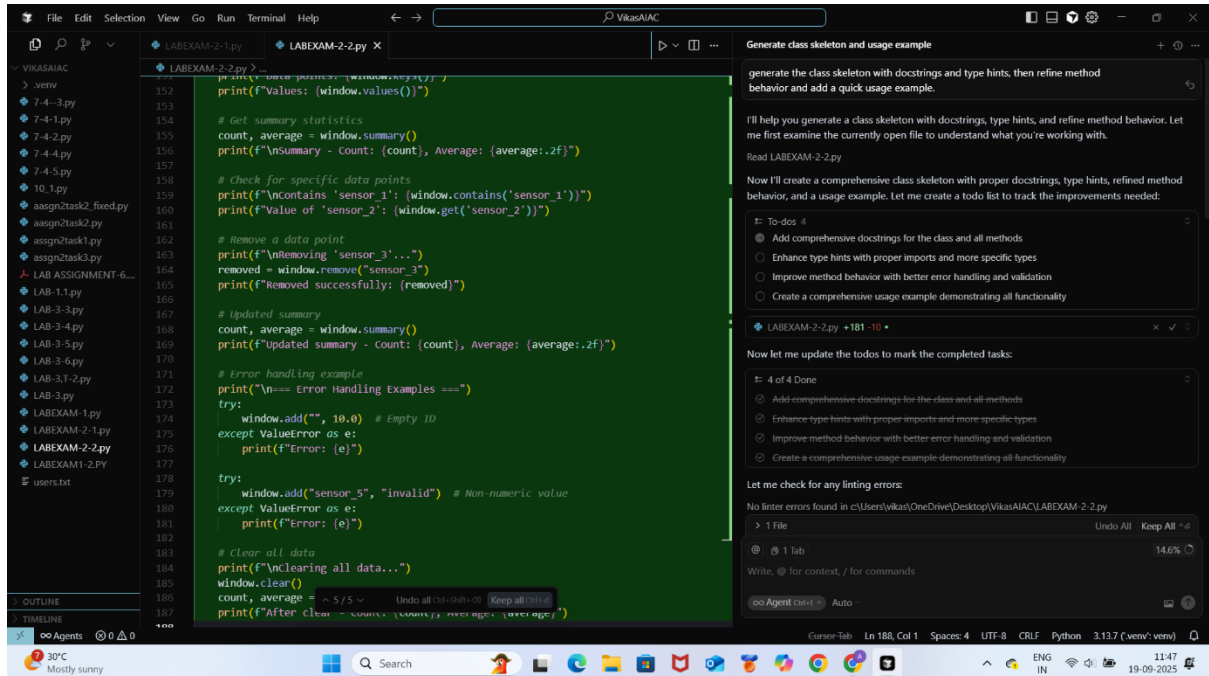
```
1 class StreamWindow:
2     def __init__(self):
3         self.data = {}
4
5     def add(self, id: str, value: float):
6         self.data[id] = value
7
8     def remove(self, id: str):
9         self.data.pop(id, None)
10
11     def summary(self) -> tuple[int, float | None]:
12         count = len(self.data)
13         if count == 0:
14             return (0, None)
15         total = sum(self.data.values())
16         return (count, total / count)
17
```

The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer lists various files including 'LABEXAM-2-1.py', 'LABEXAM-2-2.py', and 'users.txt'. The main editor window displays the Python code for the 'StreamWindow' class. The code includes methods for adding, removing, and summarizing data. The terminal at the bottom shows the status of the environment, including the Python version (3.13.7) and the current directory ('venv').

# AI – ASSISTED CODING

## LAB EXAM - 2

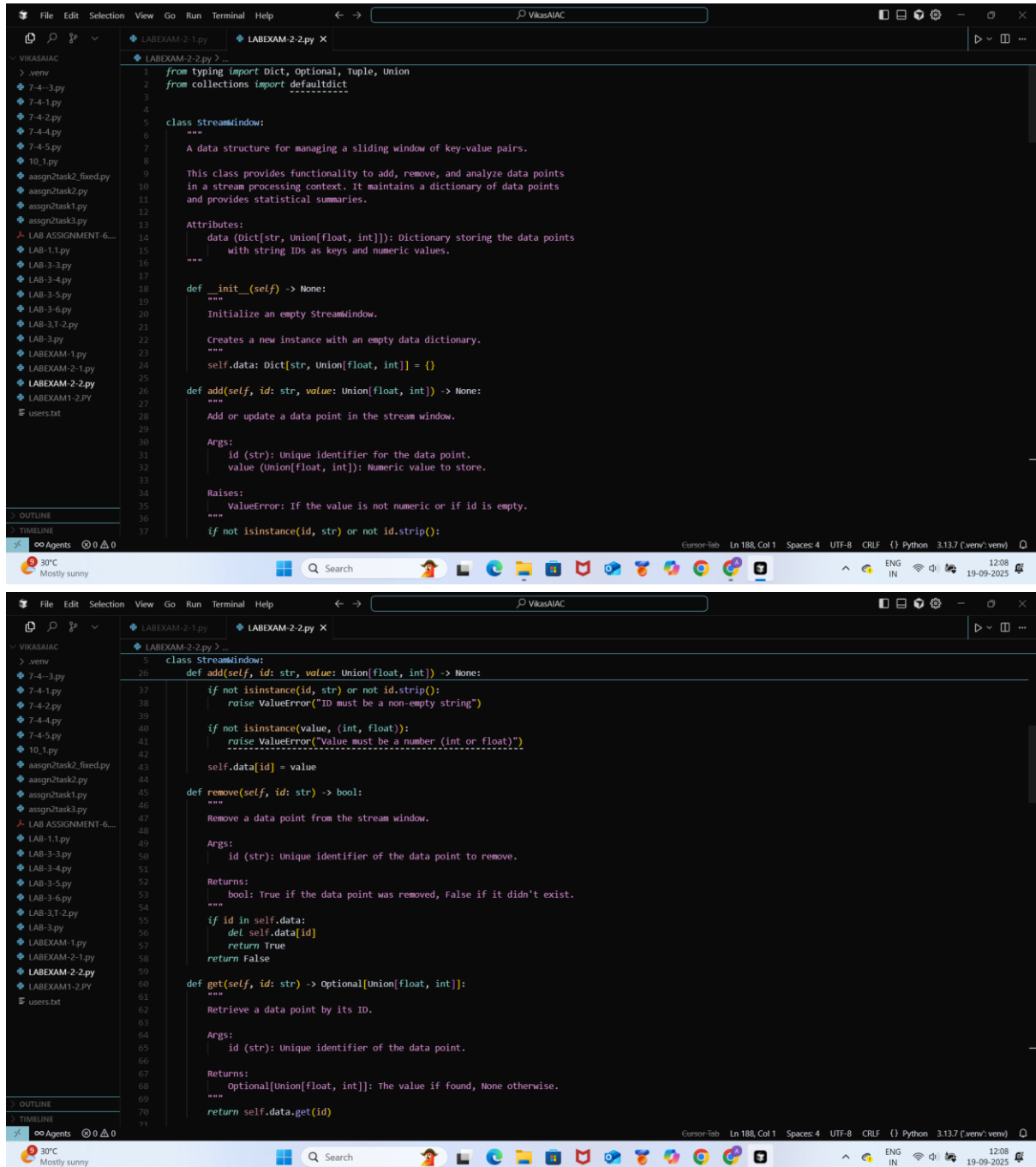
### PROMPT:



# AI – ASSISTED CODING

## LAB EXAM - 2

### AI – GENERATED CODE:



The image displays two screenshots of a code editor (VS Code) showing AI-generated Python code for a class named `StreamWindow`. The code is written in a dark theme.

**Top Screenshot:** Shows the initial part of the code. The class `StreamWindow` is defined with a docstring describing its purpose: "A data structure for managing a sliding window of key-value pairs. This class provides functionality to add, remove, and analyze data points in a stream processing context. It maintains a dictionary of data points and provides statistical summaries." The attributes section indicates that `data` is a `Dict[str, Union[float, int]]` storing data points with string IDs as keys and numeric values. The `__init__` method initializes an empty `StreamWindow` by creating a new instance with an empty data dictionary. The `add` method is partially visible, taking `self`, `id` (a string), and `value` (a float or int) as arguments.

```
1 from typing import Dict, Optional, Tuple, Union
2 from collections import defaultdict
3
4
5 class StreamWindow:
6     """
7     A data structure for managing a sliding window of key-value pairs.
8
9     This class provides functionality to add, remove, and analyze data points
10    in a stream processing context. It maintains a dictionary of data points
11    and provides statistical summaries.
12
13    Attributes:
14        data (Dict[str, Union[float, int]]): Dictionary storing the data points
15        with string IDs as keys and numeric values.
16    """
17
18    def __init__(self) -> None:
19        """
20        Initialize an empty StreamWindow.
21
22        Creates a new instance with an empty data dictionary.
23        """
24        self.data: Dict[str, Union[float, int]] = {}
25
26    def add(self, id: str, value: Union[float, int]) -> None:
27        """
28        Add or update a data point in the stream window.
29
30        Args:
31            id (str): Unique identifier for the data point.
32            value (Union[float, int]): Numeric value to store.
33
34        Raises:
35            ValueError: If the value is not numeric or if id is empty.
36        """
37        if not isinstance(id, str) or not id.strip():
```

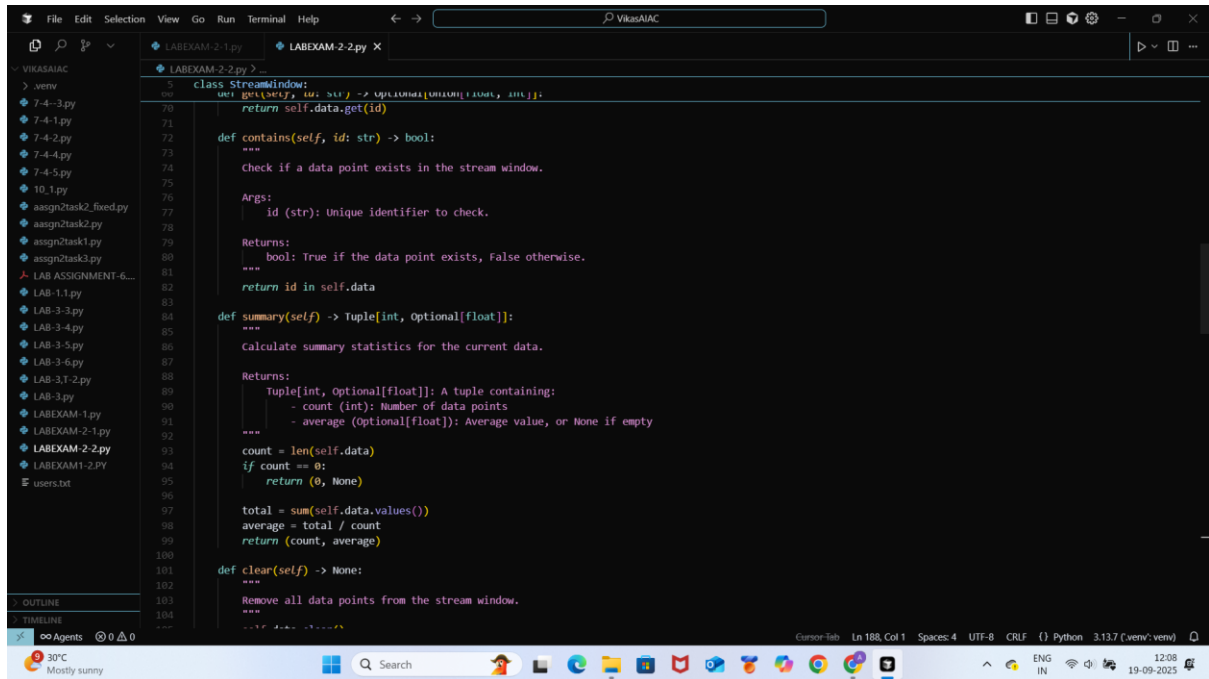
**Bottom Screenshot:** Shows the continuation of the `add` method and the `remove` and `get` methods. The `add` method includes type checking: it raises a `ValueError` if the `id` is not a non-empty string, and another `ValueError` if the `value` is not a number (int or float). It then adds the `value` to `self.data[id]`. The `remove` method takes an `id` (string) and returns a boolean, indicating whether the data point was successfully removed. The `get` method takes an `id` (string) and returns an `Optional[Union[float, int]]`, retrieving the data point by its ID.

```
38         raise ValueError("ID must be a non-empty string")
39
40         if not isinstance(value, (int, float)):
41             raise ValueError("Value must be a number (int or float)")
42
43         self.data[id] = value
44
45    def remove(self, id: str) -> bool:
46        """
47        Remove a data point from the stream window.
48
49        Args:
50            id (str): Unique identifier of the data point to remove.
51
52        Returns:
53            bool: True if the data point was removed, False if it didn't exist.
54        """
55        if id in self.data:
56            del self.data[id]
57            return True
58        return False
59
60    def get(self, id: str) -> Optional[Union[float, int]]:
61        """
62        Retrieve a data point by its ID.
63
64        Args:
65            id (str): Unique identifier of the data point.
66
67        Returns:
68            Optional[Union[float, int]]: The value if found, None otherwise.
69        """
70        return self.data.get(id)
```



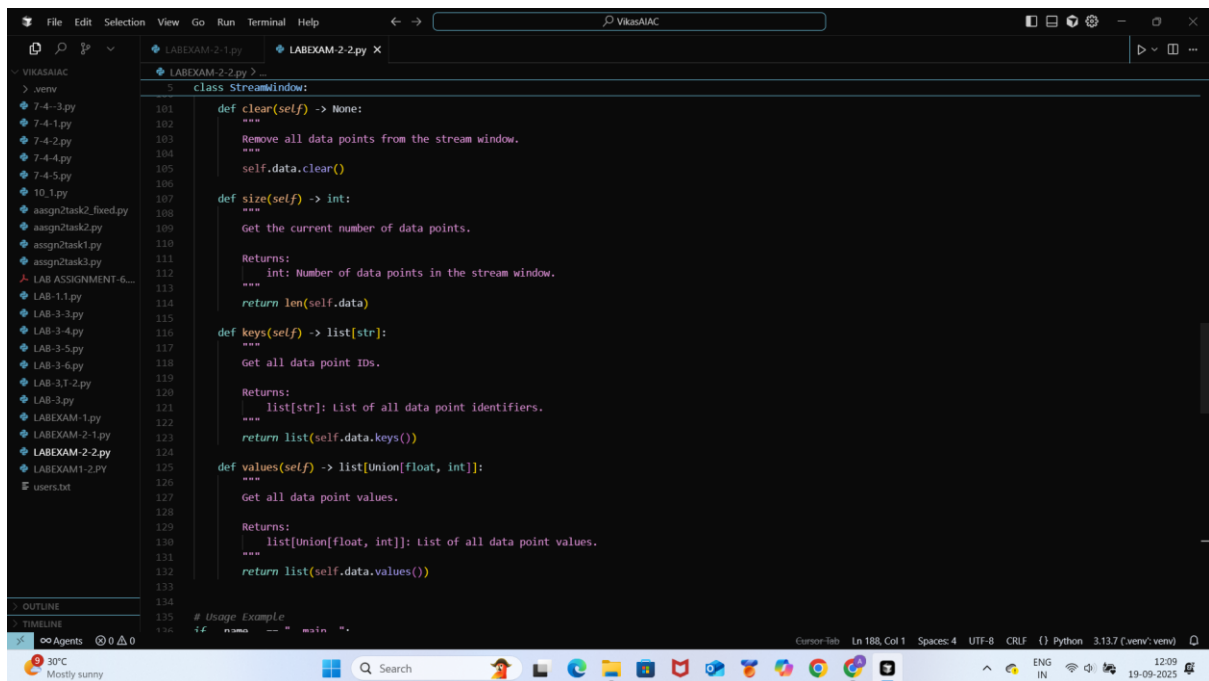
# AI – ASSISTED CODING

## LAB EXAM - 2



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'VikasAIAC' with several files including '7-4-3.py', '7-4-1.py', '7-4-2.py', '7-4-4.py', '7-4-5.py', '10\_1.py', 'aasn2task2\_fixed.py', 'aasn2task2.py', 'aasn2task1.py', 'aasn2task3.py', 'LAB ASSIGNMENT-6...', 'LAB-1.1.py', 'LAB-3-3.py', 'LAB-3-4.py', 'LAB-3-5.py', 'LAB-3-6.py', 'LAB-3.T-2.py', 'LAB-3.py', 'LABEXAM-1.py', 'LABEXAM-2-1.py', 'LABEXAM-2-2.py', 'LABEXAM1-2.PY', and 'users.txt'. The code editor shows the following Python code:

```
5 class StreamWindow:
6     def __init__(self, id: str) -> Optional[Union[float, int]]:
7         return self.data.get(id)
8
9     def contains(self, id: str) -> bool:
10        """
11        Check if a data point exists in the stream window.
12
13        Args:
14            id (str): Unique identifier to check.
15
16        Returns:
17            bool: True if the data point exists, False otherwise.
18        """
19        return id in self.data
20
21    def summary(self) -> Tuple[int, Optional[float]]:
22        """
23        Calculate summary statistics for the current data.
24
25        Returns:
26            Tuple[int, Optional[float]]: A tuple containing:
27                - count (int): Number of data points
28                - average (Optional[float]): Average value, or None if empty
29        """
30        count = len(self.data)
31        if count == 0:
32            return (0, None)
33
34        total = sum(self.data.values())
35        average = total / count
36        return (count, average)
37
38    def clear(self) -> None:
39        """
40        Remove all data points from the stream window.
41        """
```



The screenshot shows the same code editor as the previous one, but with the second part of the Python class StreamWindow. The code is as follows:

```
101    def clear(self) -> None:
102        """
103        Remove all data points from the stream window.
104        """
105        self.data.clear()
106
107    def size(self) -> int:
108        """
109        Get the current number of data points.
110
111        Returns:
112            int: Number of data points in the stream window.
113        """
114        return len(self.data)
115
116    def keys(self) -> list[str]:
117        """
118        Get all data point IDs.
119
120        Returns:
121            list[str]: List of all data point identifiers.
122        """
123        return list(self.data.keys())
124
125    def values(self) -> list[Union[float, int]]:
126        """
127        Get all data point values.
128
129        Returns:
130            list[Union[float, int]]: List of all data point values.
131        """
132        return list(self.data.values())
133
134
135 # Usage Example
136 if __name__ == '__main__':
```

# AI – ASSISTED CODING

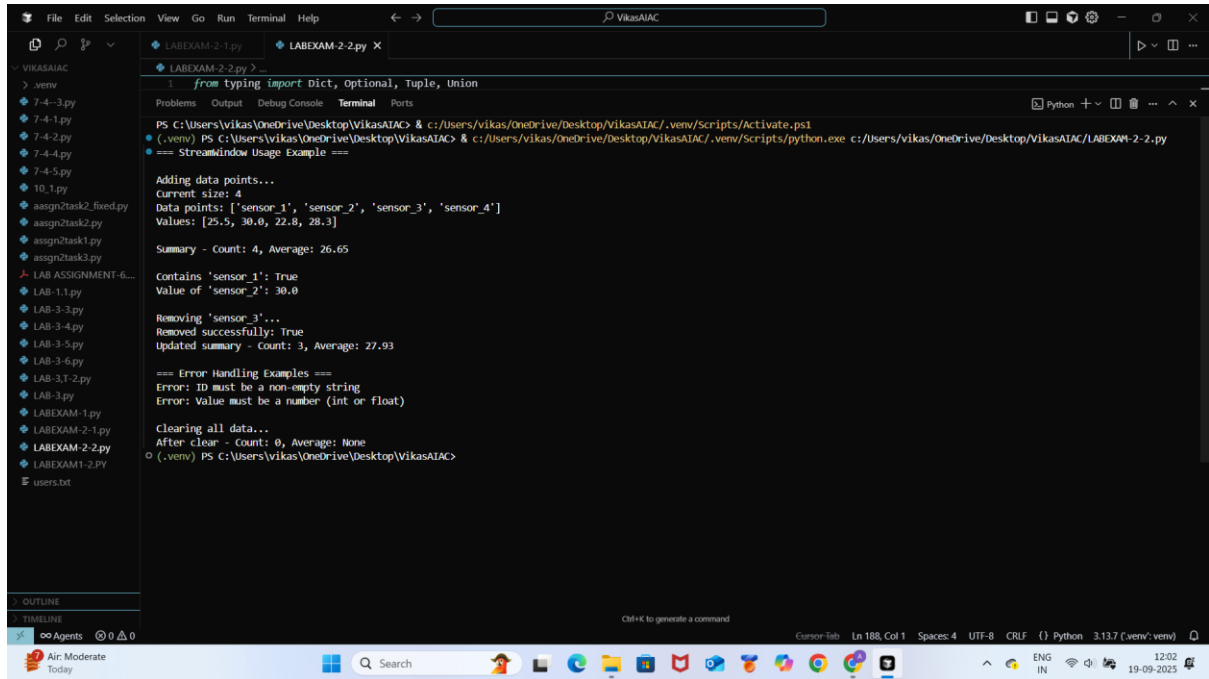
## LAB EXAM - 2

```
134
135 # Usage Example
136 if __name__ == "__main__":
137     # Create a new stream window
138     window = StreamWindow()
139
140     print("=== StreamWindow Usage Example ===\n")
141
142     # Add some data points
143     print("Adding data points...")
144     window.add("sensor_1", 25.5)
145     window.add("sensor_2", 30.0)
146     window.add("sensor_3", 22.8)
147     window.add("sensor_4", 28.3)
148
149     # Display current state
150     print(f"Current size: {window.size()}")
151     print(f"Data points: {window.keys()}")
152     print(f"Values: {window.values()}")
153
154     # Get summary statistics
155     count, average = window.summary()
156     print(f"\nSummary - Count: {count}, Average: {average:.2f}")
157
158     # Check for specific data points
159     print(f"\nContains 'sensor_1': {window.contains('sensor_1')}")
160     print(f"Value of 'sensor_2': {window.get('sensor_2')}")
161
162     # Remove a data point
163     print(f"\nRemoving 'sensor_3'...")
164     removed = window.remove("sensor_3")
165     print(f"Removed successfully: {removed}")
166
167     # Updated summary
168     count, average = window.summary()
169     print(f"Updated summary - Count: {count}, Average: {average:.2f}")
170
171
172     # Error handling example
173     print("\n=== Error Handling Examples ===")
174     try:
175         window.add("", 10.0) # Empty ID
176     except ValueError as e:
177         print(f"Error: {e}")
178
179     try:
180         window.add("sensor_5", "invalid") # Non-numeric value
181     except ValueError as e:
182         print(f"Error: {e}")
183
184     # Clear all data
185     print(f"\nClearing all data...")
186     window.clear()
187     count, average = window.summary()
188     print(f"After clear - Count: {count}, Average: {average:.2f}")
```

# AI – ASSISTED CODING

## LAB EXAM - 2

### OUTPUT:



The screenshot shows a VS Code editor with a file explorer on the left containing various Python files. The main editor window displays a Python script with the following content:

```
from typing import Dict, Optional, Tuple, Union

def main():
    # Adding data points...
    current_size = 4
    data_points = ['sensor_1', 'sensor_2', 'sensor_3', 'sensor_4']
    values = [25.5, 30.0, 22.8, 28.3]

    # Summary - Count: 4, Average: 26.65

    # Removing 'sensor_3'...
    removed_successfully = True
    updated_summary = Count: 3, Average: 27.93

    # Error Handling Examples
    error_id = ''
    error_value = 0

    # Clearing all data...
    after_clear = Count: 0, Average: None

if __name__ == '__main__':
    main()
```

The terminal window at the bottom shows the execution of the script, displaying the following output:

```
PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC> & c:/Users/vikas/OneDrive/Desktop/VikasAIAC/.venv/scripts/activate.ps1
(.venv) PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC> & c:/Users/vikas/OneDrive/Desktop/VikasAIAC/.venv/Scripts/python.exe c:/Users/vikas/OneDrive/Desktop/VikasAIAC/LABEXAM-2-2.py
StreamWindow Usage Example

Adding data points...
Current size: 4
Data points: ['sensor_1', 'sensor_2', 'sensor_3', 'sensor_4']
Values: [25.5, 30.0, 22.8, 28.3]

Summary - Count: 4, Average: 26.65

Contains 'sensor_1': True
Value of 'sensor_2': 30.0

Removing 'sensor_3'...
Removed successfully: True
Updated summary - Count: 3, Average: 27.93

Error Handling Examples
Error: id must be a non-empty string
Error: Value must be a number (int or float)

Clearing all data...
After clear - Count: 0, Average: None
(.venv) PS C:\Users\vikas\OneDrive\Desktop\VikasAIAC>
```