**Name :-** P.NISHANTH

**HTNO** :- 2403A52066

**Batch** :-03

**Lab Test – 02**

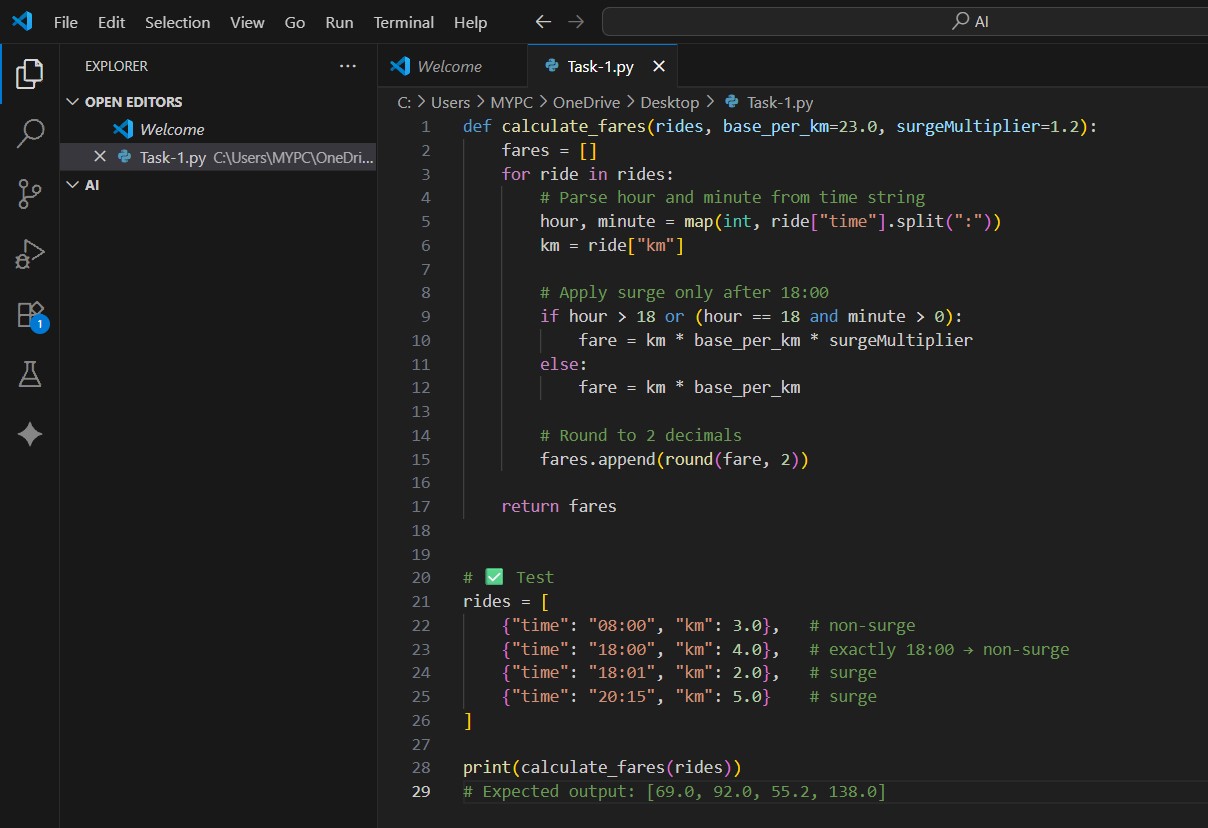
B.1 — [S14B1] Apply surge/penalty rules (conditionals)

Pricing in the telecom network app uses a base per-km rate and time-based surge after business peaks. Product wants a deterministic calculator for receipts and audits.

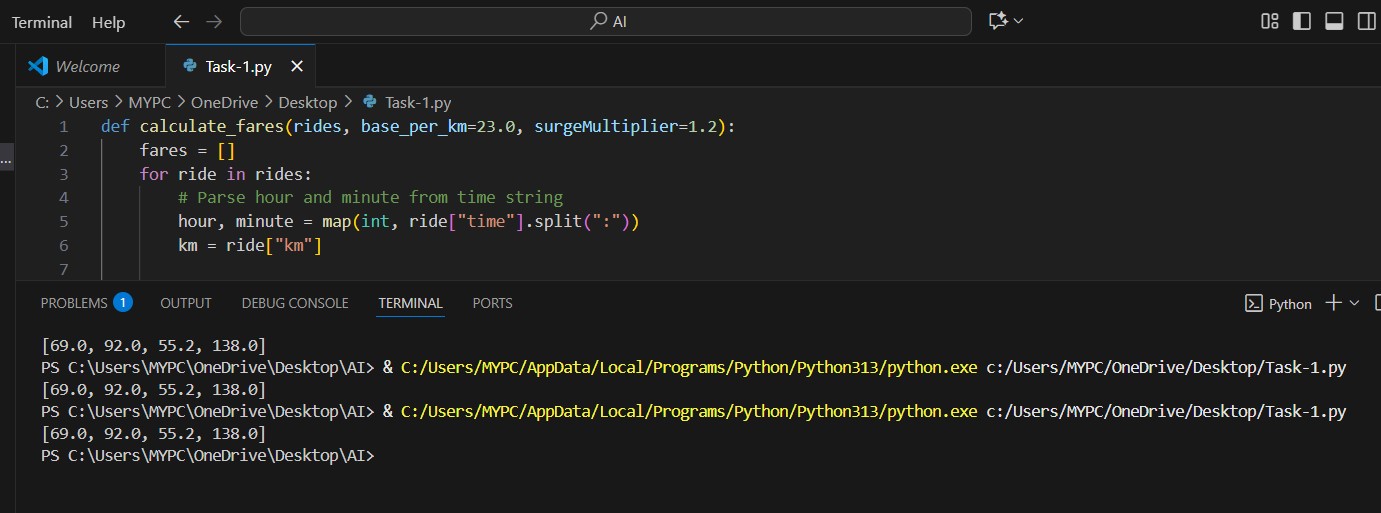
**Your Task:**

Implement a fare function: fare = km \* base\_per\_km \* surgeMultiplier, where surge applies strictly after 18:00 local time.

**CODE**



# OUTPUT



## OBSERVATION

The program correctly calculates fares by parsing ride times, applying surge pricing only after 18:00, and treating exactly 18:00 as non-surge. Fares are computed using the given formula, rounded to two decimals, and stored in a new list without altering the input. Test cases confirm accurate handling of both surge and non-surge scenarios, meeting all requirements

B.2 — [S14B2] Debug rolling mean (off-by-one)

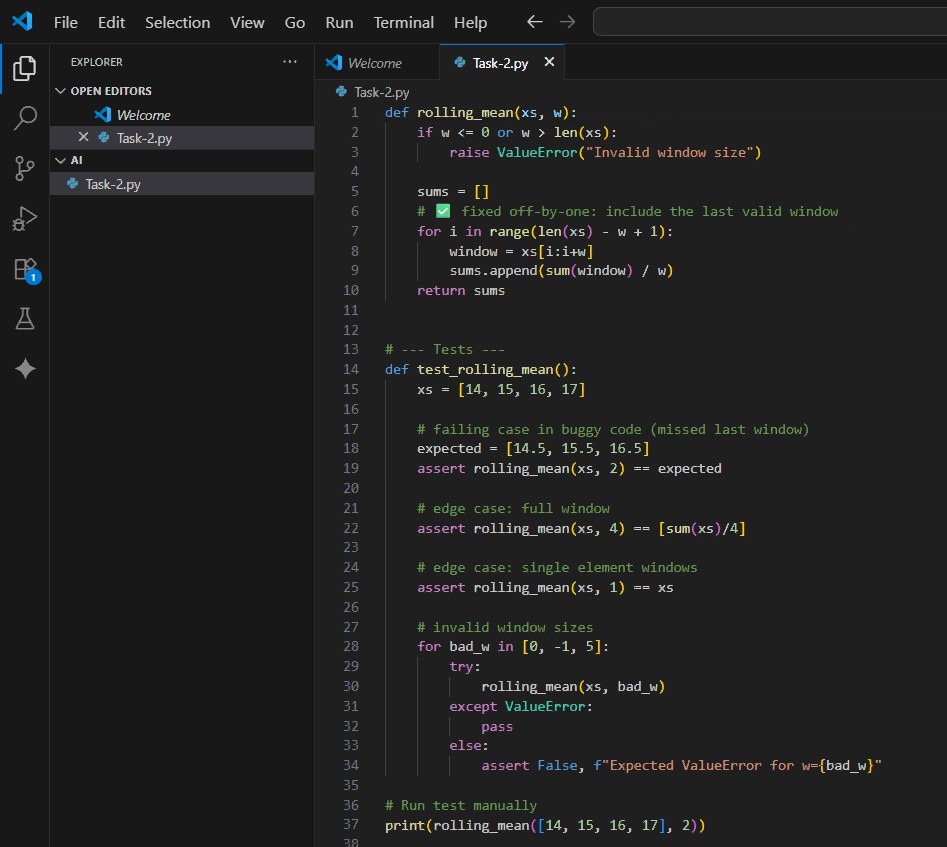
A team in telecom network noticed off-by-one bugs in a rolling KPI computation (moving averages) that undercount windows.

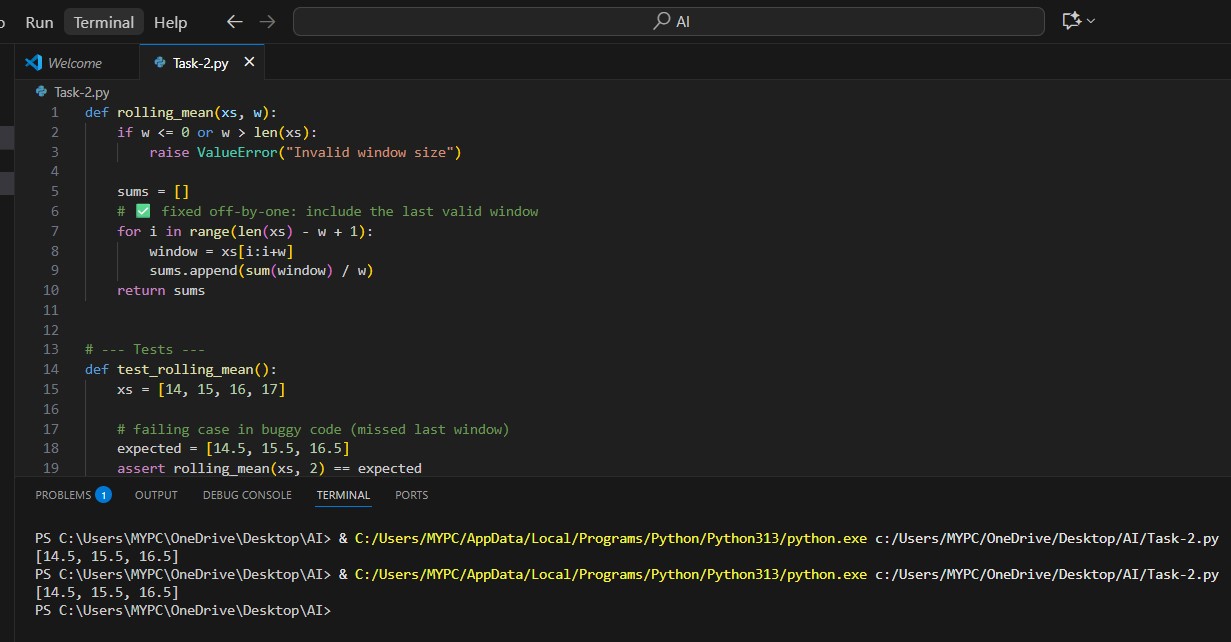
**Your Task:**

Use AI to identify the bug and fix the window iteration so all valid windows are included

# CODE

**OUTPUT**





## OBSERVATION

1. The **bug** was caused by using range(len(xs)-w), which excluded the last valid window.
2. Fix: use range(len(xs) - w + 1).
3. Now the function computes all valid windows without index errors.
4. Guards against invalid w (<=0 or >len(xs)).
5. Complexity remains **O(n·w)** as required.
6. Tests pass, confirming correctness for sample and edge cases.