Please write a Python script to solve this problem:

Certainly! Please provide the problem details so I can write the Python script for you.

Fortunately, the first location The Historians want to search isn't a long walk from the Chief Historian's office.

While the Red-Nosed Reindeer nuclear fusion/fission plant appears to contain no sign of the Chief Historian, the engineers there run up to you as soon as they see you. Apparently, they still talk about the time Rudolph was saved through molecular synthesis from a single electron.

They're quick to add that - since you're already here - they'd really appreciate your help analyzing some unusual data from the Red-Nosed reactor. You turn to check if The Historians are waiting for you, but they seem to have already divided into groups that are currently searching every corner of the facility. You offer to help with the unusual data.

The unusual data (your puzzle input) consists of many reports, one report per line. Each report is a list of numbers called levels that are separated by spaces. For example:

7 6 4 2 1
1 2 7 8 9
9 7 6 2 1
1 3 2 4 5
8 6 4 4 1
1 3 6 7 9
This example data contains six reports each containing five levels.

The engineers are trying to figure out which reports are safe. The Red-Nosed reactor safety systems can only tolerate levels that are either gradually increasing or gradually decreasing. So, a report only counts as safe if both of the following are true:

The levels are either all increasing or all decreasing.
Any two adjacent levels differ by at least one and at most three.
In the example above, the reports can be found safe or unsafe by checking those rules:

7 6 4 2 1: Safe because the levels are all decreasing by 1 or 2.
1 2 7 8 9: Unsafe because 2 7 is an increase of 5.
9 7 6 2 1: Unsafe because 6 2 is a decrease of 4.

1 3 2 4 5: Unsafe because 1 3 is increasing but 3 2 is decreasing.
8 6 4 4 1: Unsafe because 4 4 is neither an increase or a decrease.
1 3 6 7 9: Safe because the levels are all increasing by 1, 2, or 3.
So, in this example, 2 reports are safe.

Analyze the unusual data from the engineers. How many reports are safe?

Here's a Python script to analyze the unusual data and count how many reports are safe based on the given criteria:

python Copy code

```python
def is_safe_report(report):
    """
    Determines if a report is safe.
    A report is safe if:
    1. Levels are either all increasing or all decreasing.
    2. Any two adjacent levels differ by at least 1 and at most 3.
    """
    # Calculate differences between consecutive levels
    differences = [report[i+1] - report[i] for i in range(len(report) - 1)]

    # Check if all differences are either positive or negative
    increasing = all(1 <= diff <= 3 for diff in differences)
    decreasing = all(-3 <= diff <= -1 for diff in differences)

    # Report is safe if it's either all increasing or all decreasing
    return increasing or decreasing


def count_safe_reports(data):
    """
    Counts the number of safe reports in the given data.
    Each report is a list of integers (levels).
    """
    safe_count = 0
    for report in data:
        if is_safe_report(report):
            safe_count += 1
    return safe_count


# Example input data
input_data = [
    [7, 6, 4, 2, 1],
    [1, 2, 7, 8, 9],
    [9, 7, 6, 2, 1],
    [1, 3, 2, 4, 5],
    [8, 6, 4, 4, 1],
    [1, 3, 6, 7, 9]
```

```
]

# Count and print the number of safe reports
safe_reports = count_safe_reports(input_data)
print(f"Number of safe reports: {safe_reports}")
```

Explanation of the Code

1. `is_safe_report(report)`: This function checks if a single report is safe.
   - It calculates the differences between consecutive levels.
   - It ensures that all differences are either within the range for increasing levels (`1 <= diff <= 3`) or for decreasing levels (`-3 <= diff <= -1`).
   - If either condition is met, the report is safe.
2. `count_safe_reports(data)`: This function iterates over all reports and counts how many are safe using the `is_safe_report` function.
3. **Input Data**: The input data is represented as a list of lists, where each inner list contains the levels of a report.
4. **Output**: The script calculates and prints the number of safe reports.

You can replace `input_data` with the actual puzzle input to analyze the unusual data provided by the engineers.

Please modify the script to read the input from file. The file has lines of space-separated integers.

Here's the modified script that reads the input from a file:

python Copy code

```python
def is_safe_report(report):
    """
    Determines if a report is safe.
    A report is safe if:
    1. Levels are either all increasing or all decreasing.
    2. Any two adjacent levels differ by at least 1 and at most 3.
    """
    # Calculate differences between consecutive levels
    differences = [report[i+1] - report[i] for i in range(len(report) - 1)]

    # Check if all differences are either positive or negative
    increasing = all(1 <= diff <= 3 for diff in differences)
    decreasing = all(-3 <= diff <= -1 for diff in differences)
```

```
    # Report is safe if it's either all increasing or all decreasing
    return increasing or decreasing


def count_safe_reports(file_path):
    """
    Reads data from a file and counts the number of safe reports.
    Each line in the file is a space-separated list of integers.
    """
    safe_count = 0
    with open(file_path, 'r') as file:
        for line in file:
            report = list(map(int, line.strip().split()))
            if is_safe_report(report):
                safe_count += 1
    return safe_count


# File path to the input data
file_path = 'input.txt'

# Count and print the number of safe reports
safe_reports = count_safe_reports(file_path)
print(f"Number of safe reports: {safe_reports}")
```

How It Works

1. **File Reading**:
   - The `count_safe_reports` function reads the file line by line.
   - Each line is split into integers using `strip()` and `split()`, and then converted into a list of integers using `map(int, ...)`.
2. **Report Safety Check**:
   - Each report is passed to `is_safe_report` to determine if it is safe.
3. **File Input**:
   - Replace `'input.txt'` with the path to your actual file containing the space-separated integers.
4. **Output**:
   - The script will output the number of safe reports.

How to Use

1. Save the input data in a file, e.g., `input.txt`. Each line should be a space-separated list of integers, one report per line.
2. Run the script. Make sure the file is in the same directory as the script or adjust the `file_path` to the correct location.
3. The output will be the count of safe reports.

Please update the script to solve part 2 of the problem:

The engineers are surprised by the low number of safe reports until they realize they forgot to tell you about the Problem Dampener.

The Problem Dampener is a reactor-mounted module that lets the reactor safety systems tolerate a single bad level in what would otherwise be a safe report. It's like the bad level never happened!

Now, the same rules apply as before, except if removing a single level from an unsafe report would make it safe, the report instead counts as safe.

More of the above example's reports are now safe:

7 6 4 2 1: Safe without removing any level.
1 2 7 8 9: Unsafe regardless of which level is removed.
9 7 6 2 1: Unsafe regardless of which level is removed.
1 3 2 4 5: Safe by removing the second level, 3.
8 6 4 4 1: Safe by removing the third level, 4.
1 3 6 7 9: Safe without removing any level.
Thanks to the Problem Dampener, 4 reports are actually safe!

Update your analysis by handling situations where the Problem Dampener can remove a single level from unsafe reports. How many reports are now safe?

Here's the updated script to handle the Problem Dampener, allowing for the removal of a single level to make an otherwise unsafe report safe:

python Copy code

```python
def is_safe_report(report):
    """
    Determines if a report is safe.
    A report is safe if:
    1. Levels are either all increasing or all decreasing.
    2. Any two adjacent levels differ by at least 1 and at most 3.
    """
    # Calculate differences between consecutive levels
    differences = [report[i+1] - report[i] for i in range(len(report) - 1)]

    # Check if all differences are either positive or negative
    increasing = all(1 <= diff <= 3 for diff in differences)
    decreasing = all(-3 <= diff <= -1 for diff in differences)

    # Report is safe if it's either all increasing or all decreasing
```

```
    return increasing or decreasing


def is_safe_with_dampener(report):
    """
    Determines if a report is safe with the Problem Dampener.
    The Problem Dampener allows removing one level to make an unsafe report safe.
    """
    if is_safe_report(report):
        return True  # Already safe without any removal

    # Try removing each level one at a time and check if the remaining report is safe
    for i in range(len(report)):
        modified_report = report[:i] + report[i+1:]  # Remove the i-th level
        if is_safe_report(modified_report):
            return True

    return False  # Not safe even with the Problem Dampener


def count_safe_reports_with_dampener(file_path):
    """
    Reads data from a file and counts the number of safe reports with the Problem
Dampener.
    Each line in the file is a space-separated list of integers.
    """
    safe_count = 0
    with open(file_path, 'r') as file:
        for line in file:
            report = list(map(int, line.strip().split()))
            if is_safe_with_dampener(report):
                safe_count += 1
    return safe_count


# File path to the input data
file_path = 'input.txt'

# Count and print the number of safe reports with the Problem Dampener
safe_reports = count_safe_reports_with_dampener(file_path)
print(f"Number of safe reports with the Problem Dampener: {safe_reports}")
```

Key Updates

1. `is_safe_with_dampener(report)`:
   - This function checks if the report is safe directly or if it can be made safe by removing one level.
   - It iterates through all possible levels, removes each one, and checks if the modified report is safe.
2. `count_safe_reports_with_dampener(file_path)`:
   - Counts reports that are either already safe or can be made safe using the Problem Dampener.

3. **File Input**:
    - Same as before, the input is read line by line, with each line representing a report.
4. **Output**:
    - The number of reports that are safe with the Problem Dampener is printed.

How to Use

1. Place the input data in `input.txt` as before.
2. Run the script. The output will now reflect the count of reports that are safe, considering the Problem Dampener.

This script ensures that all cases, including those requiring the removal of a single level, are correctly analyzed.