AI ASSISTED CODING

LAB TEST 2

Roll no: 2403A510D1

Name: Likhitha Pothunuri

Batch 05

CSE 2nd year

SET D

D.1

Prompt:

Task: Implement Version Bumping for Sports Analytics Data Pipeline

Objective:
Create a function bump_version(name) that adds or increments a _vNN suffix to a filename with zero-padding, while preserving the original extension. If the file already has a version suffix, it should be incremented by 1; if it doesn't have one, it should be added with _v01. The version number should be zero-padded to two digits.

Constraints:

- Ensure that the original file extension is preserved.
- The version number should be zero-padded (e.g., _v01, _v02, etc.).
- Handle filenames both with and without an existing _vNN version suffix.

Code Generated:

d1.py > ...

```python
import os
import re

_VERSION_RE = re.compile(r"^(?P<prefix>.*)_v(?P<num>\d+)$")

def bump_version(name: str) -> str:
    """Return filename with bumped _vNN version suffix.

    - If `name` already has a `_vNN` (digits) suffix before the extension,
      increment the number by 1 (keeping zero-padding to at least two digits).
    - Otherwise, add `_v01` before the extension.

    The file extension (the part after the last dot) is preserved. If there
    is no extension, the suffix is appended to the full name.

    Examples:
      bump_version('data.csv') -> 'data_v01.csv'
      bump_version('report_v01.csv') -> 'report_v02.csv'
      bump_version('archive.tar.gz') -> 'archive.tar_v01.gz'
    """
    if not isinstance(name, str):
        raise TypeError("name must be a string")

    root, ext = os.path.splitext(name)

    m = _VERSION_RE.match(root)
    if m:
        prefix = m.group('prefix')
        num_str = m.group('num')
        # increment, and preserve the original zero-padding width
        num = int(num_str) + 1
        width = max(2, len(num_str))
        new_root = f"{prefix}_v{str(num).zfill(width)}"
    else:
        new_root = f"{root}_v01"

    return new_root + ext


def main():
    print("Enter a filename to bump its version suffix (e.g. 'report_v01.csv'):")
    filename = input().strip()

    try:
        bumped = bump_version(filename)
        print(f"Bumped filename: {bumped}")
    except Exception as e:
        print(f"Error: {e}")

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

Output:

```
Enter a filename to bump its version suffix (e.g. 'report_v01.csv'):
report_v01.csv
Enter a filename to bump its version suffix (e.g. 'report_v01.csv'):
report_v01.csv
report_v01.csv
Bumped filename: report_v02.csv
PS C:\Users\kavati dikshitha\OneDrive\Desktop\ai_labtest2>
```

Observation:

The bump_version(name) function is well-designed to handle versioning of filenames in a **sports analytics data pipeline**.

It correctly **adds or increments** a _vNN suffix, ensuring:

- **Zero-padding** to at least 2 digits.
- **Preservation of the original file extension**, including complex ones like .tar.gz.
- **Files with no version**
- **Files with existing single- and multi-digit versions**
- **Files without extensions**
- **Filenames with multiple dots**

D2

Prompt: Generate docstrings and usage examples

Task:

Implement a function normalize(scores) that transforms a list of numerical scores into the 0,10, 10,1 range using min-max normalization. Add:

- A **Google-style docstring** with Args, Returns, and Examples
- **Edge-case handling** for:
    - **Empty lists** → return []
    - **All scores equal (max == min)** → return list of 0.0 of same length to avoid division by zero

Improvements Required:

1. Add a **Google-style docstring** explaining:
    - What the function does
    - Parameters (Args)
    - Return value (Returns)
    - Example usage (Examples)
2. Add logic to prevent divide-by-zero error when all scores are equal (e.g., [5, 5, 5])
3. Handle empty list input

Code Generated:

d2.py > ...

```python
def normalize(scores: list[float]) -> list[float]:
    """Normalizes a list of scores to the [0, 1] range using min-max normalization.

    If all scores are equal, returns a list of 0.0s.
    If the list is empty, returns an empty list.

    Args:
        scores (list of float): A list of numerical scores.

    Returns:
        list of float: The normalized scores in the range [0, 1].

    Examples:
        >>> normalize([10, 20, 30])
        [0.0, 0.5, 1.0]

        >>> normalize([5, 5, 5])
        [0.0, 0.0, 0.0]

        >>> normalize([])
        []
    """
    if not scores:
        return []

    m = max(scores)
    n = min(scores)

    if m == n:
        return [0.0] * len(scores)

    return [(x - n) / (m - n) for x in scores]


def main():
    print("Enter your scores separated by spaces (e.g. 10 20 30):")
    user_input = input()
```

```python
def main():
    print("Enter your scores separated by spaces (e.g. 10 20 30):")
    user_input = input()

    try:
        # Convert input string into a list of floats
        scores = list(map(float, user_input.strip().split()))
    except ValueError:
        print("Invalid input! Please enter numeric scores separated by spaces.")
        return

    normalized_scores = normalize(scores)
    print("Normalized scores:")
    print(normalized_scores)


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

Output:

```
Enter your scores separated by spaces (e.g. 10 20 30):
15 30 45
Normalized scores:
shitha\OneDrive\Desktop\ai_labtest2\d2.py'
Enter your scores separated by spaces (e.g. 10 20 30):
15 30 45
Normalized scores:
Normalized scores:
[0.0, 0.5, 1.0]
PS C:\Users\kavati dikshitha\OneDrive\Desktop\ai_labtest2>
```

Observation:

The normalize function correctly implements min-max normalization, scaling scores to the 0,10,10,1 range.

It handles **edge cases** effectively:

- Returns an empty list when given an empty input.
- When all scores are identical (max == min), it returns a list of zeros, thus avoiding divide-by-zero errors.

The function includes a clear **Google-style docstring** with:

- Args describing the input parameter,
- Returns specifying the output format,
- Examples demonstrating typical and edge cases.

The interactive portion allows users to input scores manually, converting the input string into a list of floats, with error handling for invalid inputs.