Machine Learning Engineer

Creative Machine Learning Test Case

Test Case 1 : Python Programming

For this Test Case, I have developed two solutions using different combinations of functions. The code can be found in the file "Task 1: Python Programming.ipynb".

Below is the documentation for both codes :

.. autofunction:: transform_numbers

This function creatively transforms a list of numbers using a combination of mathematical operations. It offers two distinct transformation options :

Solution 1 :

**** power_and_exp : **** Applies a combination of square root and exponential functions.

Solution 2 :

****abs_and_log : **** Applies a combination of absolute value and natural logarithm functions.

****Parameters : ** (Same for both solutions)**

**** num_list` (list[float]) :** A list of numbers to be transformed.

****Returns : ** (Same for both solutions)**

*** list[float] :** A new list containing the transformed numbers. In case of errors or non-numeric elements, the corresponding element in the output list will be `None`.

****Raises : ** (Same for both solutions)**

*** ValueError :** If the input list is empty.

****Transformation Options : ****

Solution 1 : **power_and_exp**

This option applies a combination of square root (`**pow(num, 0.5)**`) and exponential (`**math.exp(num)**`) functions to each number :

*** ** Square Root (`pow(num, 0.5)`) ** :** Reduces the influence of larger numbers in the output compared to their original values.

*** **Exponential (`math.exp(num)`)** :** Introduces exponential growth for positive numbers, potentially amplifying their impact in the output.

**** Overall Effect : ****

***** Emphasises positive values due to the exponential growth.
***** Potentially compresses negative values due to the square root reduction, although very large negative numbers might still have significant impact.

Solution 2 : ** abs_and_log **

This option applies a combination of absolute value (`**abs(num)**`) and natural logarithm (`**math.log(abs(num))**`) functions to each number:

* ** **Absolute Value (`abs(num)`) ** :** Ensures we deal with positive values for the logarithm function.

* **Natural Logarithm (`math.log(abs(num))`) ** :** Introduces a logarithmic scale, compressing larger values more than smaller ones.

**** Overall Effect : ****

* Handles both positive and negative numbers consistently using the logarithm.
* Introduces a logarithmic scale, creating a more balanced representation, especially for a wide range of values.
* Gracefully handles zero input due to the absolute value.

**** Choosing the Right Option : ****

* We use `**power_and_exp**` when we want to emphasise positive values and potentially compress negative values.
* We use `**abs_and_log**` when we need to handle both positive and negative numbers and represent a wide range of values with a compressed scale.

**** Error Handling : ****

The function handles various errors gracefully:

* Empty input lists raise a `**ValueError**`.
* Non-numeric elements in the input list result in `**None**` for the corresponding position in the output list.

* The `**power_and_exp**` version handles potential `**OverflowError**` from large exponentials by returning `**None**` for that element.

* The `**abs_and_log**` version handles potential `**ValueError**` or `**ZeroDivisionError**` for invalid log arguments (e.g., negative numbers) or division by zero by returning `**None**` for that element.

**** Example Usage : ****

Refer to the provided code for examples of how to use the function with various inputs and how it handles different scenarios.