# User

bing ai

# Bing

With our revolutionary new search engine, you'll get the answers you've been looking for! We've taken Microsoft Bing's rich results and made them even better with powerful AI.

# User

Please convert following python script to xlwings udf - Please remember input as list of list and output as list of list. Input will be list of days and output will be string of aging bucket. Decorate with @xw.func and @xw.args:

import pandas as pd
pd.set\_option('display.max\_rows', None)
import sys

def aging\_buckets(x):
aging\_dict = {
range(-sys.maxsize, 0): "Advance",
range(0, 31): "Not Due",
range(31, 61): "31-60 Days",
range(61, 91): "61-90 Days",
range(91, 181): "91-180 Days",
range(181, 366): "6 Months to 1 year",
range(366, sys.maxsize): "More than 1 year"
}
for r, value in aging\_dict.items():
if x in r:
return value

df = pd.read\_csv('d:\dev\days.txt', header=None, names=['Days'])
df['Aging Buckets'] = df['Days'].apply(aging\_buckets)
print(df)

# Bing

Sure, I can help you convert your Python script to an xlwings UDF. Here's how you can do it:

import xlwings as xw

import sys

@xw.func

@xw.arg('x', ndim=2)

def aging\_buckets(x):

aging\_dict = {

range(-sys.maxsize, 0): "Advance",

range(0, 31): "Not Due",

range(31, 61): "31-60 Days",

range(61, 91): "61-90 Days",

range(91, 181): "91-180 Days",

range(181, 366): "6 Months to 1 year",

range(366, sys.maxsize): "More than 1 year"

}

result = []

for row in x:

temp = []

for item in row:

for r, value in aging\_dict.items():

if item in r:

temp.append(value)

result.append(temp)

return result

This function will take a list of lists as input (where each inner list represents a row of data), and it will return a list of lists as output. Each inner list in the output will contain the aging bucket corresponding to each day in the input.

Please note that this function is meant to be used within Excel via xlwings. It won't work if you try to run it in a standalone Python script. Also, you need to have the xlwings add-in installed and activated in Excel. If you haven't done so already, you can follow the instructions in the [xlwings documentation](https://docs.xlwings.org/en/stable/addin.html) to set it up. Once you've done that, you can call this function directly from your Excel workbook. For example, if you have a range of cells in Excel with days, you can use =aging\_buckets(A1:B10) to get the aging buckets for those days. The range A1:B10 is just an example, you should replace it with the actual range of cells that contain your data.

I hope this helps! Let me know if you have any other questions. 😊

# User

Function just hanged. Python was running in the background so some processing was going on. Why this is too slow? My excel range has 10000 datapoints. Please check, if there is any method to speed up the code execution.

# Bing

The function might be slow due to the nested loops and the in operator used in the function. The in operator in Python has a time complexity of O(n) and when used inside a loop, it can lead to a time complexity of O(n^2), which can be quite slow for large datasets.

To speed up the function, we can change the data structure used for aging\_dict. Instead of using range objects as keys, which requires the in operator to iterate over each element in the range, we can use a list of tuples, where each tuple represents an interval. Then, we can use a binary search to find the interval that contains item, which is much faster. Here's how you can modify the function:

import xlwings as xw

import sys

import bisect

@xw.func

@xw.arg('x', ndim=2)

def aging\_buckets(x):

aging\_list = [

(-sys.maxsize, 0, "Advance"),

(0, 31, "Not Due"),

(31, 61, "31-60 Days"),

(61, 91, "61-90 Days"),

(91, 181, "91-180 Days"),

(181, 366, "6 Months to 1 year"),

(366, sys.maxsize, "More than 1 year")

]

result = []

for row in x:

temp = []

for item in row:

index = bisect.bisect(aging\_list, (item,))

if index:

\_, \_, value = aging\_list[index - 1]

temp.append(value)

result.append(temp)

return result

This function should be significantly faster for large datasets. Please give it a try and let me know if it helps! 😊

# User

That did the trick! I think that bisect option! Seems to be a built in library of python.

# Bing

Yes, you're correct! The bisect module is a built-in library in Python. It provides support for maintaining a list in sorted order without having to sort the list after each insertion. For certain kinds of computations, this can be much more efficient than sorting at each insertion. I'm glad to hear that the solution worked for you! If you have any more questions or need further assistance, feel free to ask. 😊