SEAS 6414

Spring 2024

Assignment 3

Dr. Adewale Akinfaderin

I am coming to terms with using exec() for the homework. There are still several minor issues. But all in all, I think it works well.

Let me know if you want to see the source file. This is the executed file and has everything in it. But I am happy to share the source file. This assignment is in GitHub at <https://github.com/OwlSaver/GWU>.

# Execution

#######################################################################################

# Problem 1

#######################################################################################

Problem:

Use NumPy to identify unique elements in an array and count their occurrences.

Problem Statement:

Given the NumPy array:

x = [3, 1, 4, 2, 4, 3, 6, 1, 2, 5, 5, 6, 2, 3]

Write a Python function using NumPy to accomplish the following tasks:

1. Extract an array of unique elements from array x.

2. Create an array representing the count of each unique element in x.

Expected Output:

For the provided array x, your function should return:

- Unique elements array: [1, 2, 3, 4, 5, 6]

- Counts array: [2, 3, 3, 2, 2, 2]

Code:

import numpy as np

def UniqueElements(anArray):

import numpy as np

ue, ca = np.unique(anArray,return\_counts = True)

print(f"Unique elements array: {ue}")

print(f"Counts array: {ca}")

x = np.array([3, 1, 4, 2, 4, 3, 6, 1, 2, 5, 5, 6, 2, 3])

UniqueElements(x)

Execution:

Unique elements array: [1 2 3 4 5 6]

Counts array: [2 3 3 2 2 2]

#######################################################################################

# Problem 2

#######################################################################################

Problem:

Generate a series of normal random variables for different sample sizes and compute

their averages.

Task:

- For each N in {5, 20, 100, 500, 2000, 50000}, generate N normal random

variables.

- Each set of random variables should have a mean of 10 and a standard deviation

of 5.

- Compute the average of these random variables for each N.

- Store the averages in a NumPy array.

- Additionally, write the results to a file using NumPy's save function.

Provide a printout of the final array. (Note: You do not need to submit the file

itself.)

Expected Output: A NumPy array containing the average values for each specified

N.

Code:

import numpy as np

S = {5, 20, 100, 500, 2000, 50000}

T = np.array([np.average(np.random.normal(10,5,N)) for N in S])

np.save('.\\HW3Output.npy',T)

print(T)

Execution:

[ 9.92240637 10.00547795 10.16160839 9.04581491 10.01050571 13.37951234]

#######################################################################################

# Problem 3

#######################################################################################

Problem:

Implement a NumPy program to pad strings with leading zeros to create a uniform

numeric string length.

Task Description:

- Given an array of string elements representing numbers, transform each element

into a 5-digit numeric string.

- Pad strings with fewer than 5 digits with leading zeros.

- Strings with 5 or more digits should remain unchanged.

Example:

- Original Array: ['2', '11', '234', '1234', '12345']

- Formatted Output: ['00002', '00011', '00234', '01234', '12345']

Implementation Requirement:

- Utilize NumPy's capabilities for efficient string manipulation and array processing.

Code:

import numpy as np

def PadTo5(anArray):

import numpy as np

mask = np.char.str\_len(anArray) < 6 # Needed because zfill will truncate everything to 5

anArray[mask] = np.char.zfill(anArray[mask], 5)

return anArray

X = np.array(['2', '11', '234', '1234', '12345'])

print(f"Original Array: {X}")

Y = PadTo5(X)

print(f"Formatted Output: {Y}")

M = np.array(['2', '11', '234', '1234', '12345', '1234567'])

print(f"Original Array: {M}")

N = PadTo5(M)

print(f"Formatted Output: {N}")

Execution:

Original Array: ['2' '11' '234' '1234' '12345']

Formatted Output: ['00002' '00011' '00234' '01234' '12345']

Original Array: ['2' '11' '234' '1234' '12345' '1234567']

Formatted Output: ['00002' '00011' '00234' '01234' '12345' '1234567']

#######################################################################################

# Problem 4

#######################################################################################

Problem:

Implement a Python function using NumPy to convert Cartesian coordinates to polar

coordinates.

Task Details:

- Generate a random 10x2 matrix using NumPy, where each row represents a

Cartesian coordinate (x, y).

- Develop a function to convert these Cartesian coordinates into polar coordinates

(r, theta).

- The polar coordinates should be calculated as follows:

- r = sqroot(x\*\*2 + y\*\*2) (radial distance)

- theta = arctan(y/x) (angle in radians)

- The function should return a new 10x2 matrix with polar coordinates.

Example: For a point (x, y) in the Cartesian coordinate system, the corresponding

polar coordinates (r, theta) should be computed and stored in the resulting matrix.

Code:

import numpy as np

def Cart2Polar(cartArray):

import numpy as np

polarArray = cartArray.copy()

for i in range(cartArray.shape[0]):

polarArray[i,0]=np.sqrt((cartArray[i,0]\*\*2) + (cartArray[i,1]\*\*2))

polarArray[i,1]=np.arctan(cartArray[i,1] / cartArray[i,0])

return polarArray

cartArray = np.random.uniform(-100,100,(10,2))

polarArray = Cart2Polar(cartArray)

np.set\_printoptions(suppress=True,precision=4)

print("Cartesian")

print(cartArray)

print("Polar")

print(polarArray)

Execution:

Cartesian

[[ 65.3408 -12.4547]

[-12.6333 -25.657 ]

[-54.4387 37.0352]

[-58.2536 63.5461]

[ 46.9209 -57.0752]

[ 28.96 99.846 ]

[ 48.9454 -9.0583]

[ 12.0659 -69.5038]

[ 56.6431 76.789 ]

[-12.4392 75.1779]]

Polar

[[ 66.5172 -0.1884]

[ 28.5986 1.1133]

[ 65.8421 -0.5974]

[ 86.2067 -0.8288]

[ 73.886 -0.8827]

[103.961 1.2885]

[ 49.7765 -0.183 ]

[ 70.5433 -1.3989]

[ 95.4201 0.9352]

[ 76.2 -1.4068]]

#######################################################################################

# Problem 5

#######################################################################################

Problem:

Manually compute the covariance matrix of two given datasets without using the

built-in 'numpy.cov' function.

Task Description:

- Given two 1D NumPy arrays x and y, representing two different datasets.

- Write a Python function using NumPy to calculate the covariance matrix of x

and y.

- The function should manually compute the covariance values, without utilizing

the 'numpy.cov' function.

- Validate your function by comparing its output with manually computed covariance values.

Covariance Formula:

- The covariance between two variables x and y can be computed as:

Cov(x,y)=E[(x-Ex)(y-Ey)]=E[xy]-(Ex)(Ey).

Expected Output:

- A 2x2 covariance matrix representing the covariance between x and y.

Code:

import numpy as np

def cov\_value(x,y):

mean\_x = sum(x) / float(len(x))

mean\_y = sum(y) / float(len(y))

sub\_x = [i - mean\_x for i in x]

sub\_y = [i - mean\_y for i in y]

sum\_value = sum([sub\_y[i]\*sub\_x[i] for i in range(len(x))])

denom = float(len(x)-1)

cov = sum\_value/denom

return cov

def covariance(x, y):

c = np.array([[cov\_value(x,x), cov\_value(y,x)], [cov\_value(x,y), cov\_value(y,y)]])

return c

np.set\_printoptions(suppress=True,precision=2)

x = np.array([1, 2, 3, 4, 5])

y = np.array([1, 1, 1, 1, 1])

print(f"The x array is {x} and the y array is {y}.")

print("Manually calculated covariance:")

print(covariance(x, y))

print("NumPy calculated covariance:")

print(np.cov(x, y))

print("")

x = np.random.uniform(-100,100,10)

y = np.random.uniform(-100,100,10)

print(f"The x array is {x} and the y array is {y}.")

print("Manually calculated covariance:")

print(covariance(x, y))

print("NumPy calculated covariance:")

print(np.cov(x, y))

Execution:

The x array is [1 2 3 4 5] and the y array is [1 1 1 1 1].

Manually calculated covariance:

[[2.5 0. ]

[0. 0. ]]

NumPy calculated covariance:

[[2.5 0. ]

[0. 0. ]]

The x array is [-14.99 57.3 -79.23 -78.61 3.78 -76.67 -64.8 30.17 53.77 -38.41] and the y array is [-60.32 -90.52 -80.19 92.92 31.46 2.43 -84.1 -33.76 -76.9 -43.03].

Manually calculated covariance:

[[ 3002.13 -1161.67]

[-1161.67 3574.57]]

NumPy calculated covariance:

[[ 3002.13 -1161.67]

[-1161.67 3574.57]]

#######################################################################################

# Problem 6

#######################################################################################

Problem:

Create a 2D matrix from a given 1D array using specific window length and strides.

Problem Statement:

Consider the following 1D NumPy array named 'arr'. Your task is to write a Python

program using NumPy to transform 'arr' into a 2D matrix. The matrix should be

constructed by applying a sliding window approach with a specified window length

and stride.

Task Details:

1. Given a 1D NumPy array 'arr'.

2. Create a 2D matrix where each row is generated by sliding a window of length

4 over 'arr'.

3. The stride for the sliding window should be 2 elements.

4. Example: If 'arr' is [0, 1, 2, 3, 4, 5, 6, 7, 8, . . .], the resulting matrix should be:

0 1 2 3

2 3 4 5

4 5 6 7

.. .. .. ..

- Provide the Python code for generating the 2D matrix from 'arr'.

Code:

import numpy as np

window = 4

stride = 2

arr = np.arange(20)

maxsteps = int((arr.shape[0] - window) / stride)

newarr = np.zeros((maxsteps,window), dtype=int)

for i in range(maxsteps):

start = i \* stride

end = start + window

newarr[i] = arr[start:end]

print("Input vector is:")

print(arr)

print("")

print(f"Using the input vector with a window of {window} and stride of {stride} results in:")

print(newarr)

Execution:

Input vector is:

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

Using the input vector with a window of 4 and stride of 2 results in:

[[ 0 1 2 3]

[ 2 3 4 5]

[ 4 5 6 7]

[ 6 7 8 9]

[ 8 9 10 11]

[10 11 12 13]

[12 13 14 15]

[14 15 16 17]]

#######################################################################################

# Problem 7

#######################################################################################

Problem:

Develop a NumPy program to compute one-hot encodings for a given array.

Problem Statement:

One-hot encoding is a process by which categorical variables are converted into a

binary (0 or 1) matrix. Your task is to write a Python function using NumPy to

create one-hot encodings for each unique value in a given array.

Task Details:

1. Given the 1D NumPy array: array([2, 3, 2, 4, 1, 2]).

2. Your function should compute the one-hot encoding for this array.

3. Each unique value in the array should correspond to a column in the resulting

binary matrix.

Example:

- Input Array: array([2, 3, 2, 4, 1, 2])

- One-Hot Encoding Output:

0 1 0 0

0 0 1 0

0 1 0 0

0 0 0 1

1 0 0 0

0 1 0 0

Submission:

- Provide the Python code for your one-hot encoding function. The ONLY library

you should import to solve this problem is Numpy.

Code:

import numpy as np

def OneHotEncoding(aVector):

cols = int(np.max(aVector) + 0.5)

rows = aVector.shape[0]

OHE = np.zeros((rows,cols), dtype=int)

for i in range(rows):

OHE[i,aVector[i] - 1] = 1

return OHE

vec = np.array([2, 3, 2, 4, 1, 2])

OHE = OneHotEncoding(vec)

print(f"The input vector is: {vec}.")

print("The one-hot encoding is:")

print(OHE)

Execution:

The input vector is: [2 3 2 4 1 2].

The one-hot encoding is:

[[0 1 0 0]

[0 0 1 0]

[0 1 0 0]

[0 0 0 1]

[1 0 0 0]

[0 1 0 0]]