Assignment1_Group23

July 25, 2021

UCLA Anderson MFE (2021-22) BootCamp

Assignment - Module 1

Authors: Aman Jindal | Yuhang Jiang | Daniel Gabriel Tan | Qining Liu

Q1. Create a list([3,4,5,7,8]) and store it in variable "my_list". The list contains 5 elements. Write a program to interchange the first and second elements with the last and second last elements respectively. The new list will be [8,7,5,4,3].

Answer 1:

[1]:
```
my_list = [3,4,5,7,8]

def swap_list_elements(list, pos1, pos2):
    list[pos1], list[pos2] = list[pos2], list[pos1]
    return list

swap_list_elements(my_list, 0, len(my_list) - 1)
swap_list_elements(my_list, 1, len(my_list) - 2)

print('The Swapped List: {}'.format(my_list))
```

```
The Swapped List: [8, 7, 5, 4, 3]
```

Q2. Create a dictionary with five stocks (any five stocks from S&P 500). The keys of the dictionary that you will have to populate are - Name, Company Sector, Price, Market Cap (USD Mn), and Price-Earnings Ratio. You should find such data on Yahoo Finance. The keys have to be strings. For example (https://finance.yahoo.com/quote/TSLA/).

Answer 2:

[2]:
```
# Importing Required Libraries

import yfinance as yf
from pprint import pprint
```

[3]:
```
# List of Stocks for which we will collect data
tickers_list = ['MSFT', 'HD', 'LOW', 'ADBE', 'DUK']

# Collating Keys in Yahoo Finance Data Corresponding to our requirements
stock_info_reqd = {'Name':'longName',
                        'Company_Sector':'sector',
                        'Price (USD)':'currentPrice',
                        'Market_Cap(USD_Million)':'marketCap',
                        'PE_Ratio':'trailingPE'}

# Keys for which value needs to be converted into million
million_conv_keys = ['Market_Cap(USD_Million)']
```

[4]:
```
# Company Class for fetching the requisite data

class Company:

    def __init__(self, ticker, stock_info_reqd, million_conv_keys):

        self.ticker = ticker
        self.internal_fetch_stock_info(ticker, stock_info_reqd,␣
↪million_conv_keys)

    def __repr__(self):
        return self.ticker

    def internal_fetch_stock_info(self, ticker, stock_info_reqd,␣
↪million_conv_keys):

        self.stock_info_dict = dict(zip(stock_info_reqd.keys(),
                            list(map(yf.Ticker(ticker).info.get,␣
↪stock_info_reqd.values()))))

        for key in million_conv_keys:
            self.stock_info_dict[key] = self.stock_info_dict[key]/10**6

    def external_fetch_stock_info(self):
        return self.stock_info_dict

    def stock_info_keys(self):
        return self.stock_info_dict.keys()
```

[5]:
```
# Creating Dictionary for the 5 stocks

companies_info_final = {ticker :
                        Company(ticker = ticker,
                        stock_info_reqd = stock_info_reqd,
```

```
                        million_conv_keys = million_conv_keys).
↪external_fetch_stock_info()
                        for ticker in tickers_list}
```

[6]:
```
# Printing the Result

print('\033[1m\t\t\033[4mDictionary of Companies\033[0m\033[0m \n\n')
pprint(companies_info_final, sort_dicts=False)
```

Dictionary of Companies

```
{'MSFT': {'Name': 'Microsoft Corporation',
          'Company_Sector': 'Technology',
          'Price (USD)': 289.67,
          'Market_Cap(USD_Million)': 2181669.978112,
          'PE_Ratio': 39.475338},
 'HD': {'Name': 'The Home Depot, Inc.',
        'Company_Sector': 'Consumer Cyclical',
        'Price (USD)': 332.84,
        'Market_Cap(USD_Million)': 353895.481344,
        'PE_Ratio': 24.263012},
 'LOW': {'Name': "Lowe's Companies, Inc.",
         'Company_Sector': 'Consumer Cyclical',
         'Price (USD)': 200.84,
         'Market_Cap(USD_Million)': 141971.78368,
         'PE_Ratio': 21.906631},
 'ADBE': {'Name': 'Adobe Inc.',
          'Company_Sector': 'Technology',
          'Price (USD)': 625.87,
          'Market_Cap(USD_Million)': 298164.453376,
          'PE_Ratio': 54.239536},
 'DUK': {'Name': 'Duke Energy Corporation',
         'Company_Sector': 'Utilities',
         'Price (USD)': 104.47,
         'Market_Cap(USD_Million)': 80360.308736,
         'PE_Ratio': 59.156284}}
```

Q3. Using NumPy's arange and linspace, generate the same arrays between [0,10] and the array has size of 101 elements. Check if all the elements match.

Answer 3:

[7]:
```
# Importing Required Libraries

import numpy as np
np.set_printoptions(precision=4) # To Print Values till 4 decimal places
```

[8]:
```
# Creating & Printing Linspace Array

start = 0
stop = 10
array_size = 101

array_linspace = np.linspace(start = start, stop = stop,
                             num = array_size, endpoint = False, dtype = float)

print('Linspace Array: \n\n {}'.format(array_linspace))
```

```
Linspace Array:

 [0.     0.099  0.198  0.297  0.396  0.495  0.5941 0.6931 0.7921 0.8911
 0.9901 1.0891 1.1881 1.2871 1.3861 1.4851 1.5842 1.6832 1.7822 1.8812
 1.9802 2.0792 2.1782 2.2772 2.3762 2.4752 2.5743 2.6733 2.7723 2.8713
 2.9703 3.0693 3.1683 3.2673 3.3663 3.4653 3.5644 3.6634 3.7624 3.8614
 3.9604 4.0594 4.1584 4.2574 4.3564 4.4554 4.5545 4.6535 4.7525 4.8515
 4.9505 5.0495 5.1485 5.2475 5.3465 5.4455 5.5446 5.6436 5.7426 5.8416
 5.9406 6.0396 6.1386 6.2376 6.3366 6.4356 6.5347 6.6337 6.7327 6.8317
 6.9307 7.0297 7.1287 7.2277 7.3267 7.4257 7.5248 7.6238 7.7228 7.8218
 7.9208 8.0198 8.1188 8.2178 8.3168 8.4158 8.5149 8.6139 8.7129 8.8119
 8.9109 9.0099 9.1089 9.2079 9.3069 9.4059 9.505  9.604  9.703  9.802
 9.901 ]
```

[9]:
```
# Creating & Printing Arange Array

start = 0
stop = 10
array_size = 101
step_size = (stop - start)/array_size

array_arange = np.arange(start = start, stop = stop, step = step_size, dtype =␣
 ↪float)

print('Arange Array: \n\n {}'.format(array_arange))
```

```
Arange Array:

 [0.     0.099  0.198  0.297  0.396  0.495  0.5941 0.6931 0.7921 0.8911
 0.9901 1.0891 1.1881 1.2871 1.3861 1.4851 1.5842 1.6832 1.7822 1.8812
 1.9802 2.0792 2.1782 2.2772 2.3762 2.4752 2.5743 2.6733 2.7723 2.8713
 2.9703 3.0693 3.1683 3.2673 3.3663 3.4653 3.5644 3.6634 3.7624 3.8614
 3.9604 4.0594 4.1584 4.2574 4.3564 4.4554 4.5545 4.6535 4.7525 4.8515
 4.9505 5.0495 5.1485 5.2475 5.3465 5.4455 5.5446 5.6436 5.7426 5.8416
 5.9406 6.0396 6.1386 6.2376 6.3366 6.4356 6.5347 6.6337 6.7327 6.8317
 6.9307 7.0297 7.1287 7.2277 7.3267 7.4257 7.5248 7.6238 7.7228 7.8218
 7.9208 8.0198 8.1188 8.2178 8.3168 8.4158 8.5149 8.6139 8.7129 8.8119
```

```
 8.9109 9.0099 9.1089 9.2079 9.3069 9.4059 9.505  9.604  9.703  9.802
 9.901 ]
```

[10]:
```
# Validating if the Linspace & Arange Arrays are Equal

print('Check if the Arrays are Equal: \n\n {}'.format(array_linspace ==␣
 ↪array_arange))
```

```
Check if the Arrays are Equal:

 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True]
```

Q4. Generate 50 random numbers from [0,1) using np.random.rand function and calculate the mean, median, and variance.

Answer 4:

[11]:
```
# Generating & Printing Uniform [0,1) random array

array_size = 50
np.random.seed(91) # Setting Seed as 91

arr_uniform_rand = np.random.rand(array_size)
print(arr_uniform_rand)
print('\nArray Dimensions: {}'.format(arr_uniform_rand.shape))
```

```
[0.201  0.329  0.2965 0.0933 0.3331 0.7252 0.6524 0.5049 0.9473 0.6274
 0.9923 0.0761 0.3426 0.3069 0.7102 0.9173 0.0809 0.0628 0.5822 0.6692
 0.0093 0.4219 0.3954 0.0732 0.7703 0.1894 0.468  0.6517 0.0082 0.068
 0.7469 0.1575 0.5348 0.6763 0.5196 0.5194 0.4658 0.2446 0.9115 0.1077
 0.4818 0.7487 0.385  0.7519 0.7596 0.2879 0.6734 0.1756 0.672  0.2965]

Array Dimensions: (50,)
```

[12]:
```
# Calculating & Printing Mean, Median & Variance

arr_uniform_rand_mean = np.mean(a = arr_uniform_rand, axis = None)
arr_uniform_rand_median = np.median(a = arr_uniform_rand, axis = None)
arr_uniform_rand_var = np.var(a = arr_uniform_rand, ddof = 0, axis = None)
```

```
print('Mean of the Array: {:.4f}'.format(arr_uniform_rand_mean))
print('Median of the Array: {:.4f}'.format(arr_uniform_rand_median))
print('Variance of the Array: {:.4f}'.format(arr_uniform_rand_var))
```

```
Mean of the Array: 0.4525
Median of the Array: 0.4669
Variance of the Array: 0.0751
```

Q5. Now reshape the above array into a 10x5 matrix (10 rows and 5 columns). calculate the same statistics for each column.

Answer 5:

[13]:
```
# Reshaping the Array to (10,5)

newshape = (10,5)

arr_uniform_rand_reshaped = np.reshape(a = arr_uniform_rand, newshape =␣
 ↪newshape, order = 'C')
print('Reshaped Array: \n\n {}'.format(arr_uniform_rand_reshaped))
print('\n Array Dimensions: {}'.format(arr_uniform_rand_reshaped.shape))
```

```
Reshaped Array:

 [[0.201  0.329  0.2965 0.0933 0.3331]
 [0.7252 0.6524 0.5049 0.9473 0.6274]
 [0.9923 0.0761 0.3426 0.3069 0.7102]
 [0.9173 0.0809 0.0628 0.5822 0.6692]
 [0.0093 0.4219 0.3954 0.0732 0.7703]
 [0.1894 0.468  0.6517 0.0082 0.068 ]
 [0.7469 0.1575 0.5348 0.6763 0.5196]
 [0.5194 0.4658 0.2446 0.9115 0.1077]
 [0.4818 0.7487 0.385  0.7519 0.7596]
 [0.2879 0.6734 0.1756 0.672  0.2965]]

 Array Dimensions: (10, 5)
```

[14]:
```
# Calculating & Printing Mean, Median & Variance Column Wise

arr_uniform_rand_reshaped_mean = np.mean(a = arr_uniform_rand_reshaped, axis =␣
 ↪0)
arr_uniform_rand_reshaped_median = np.median(a = arr_uniform_rand_reshaped,␣
 ↪axis = 0)
arr_uniform_rand_reshaped_var = np.var(a = arr_uniform_rand_reshaped, ddof = 0,␣
 ↪axis = 0)
```

```
print('Means of the Array Column Wise: {}'.
 ↪format(arr_uniform_rand_reshaped_mean))
print('Medians of the Array Column Wise: {}'.
 ↪format(arr_uniform_rand_reshaped_median))
print('Variances of the Array Column Wise: {}'.
 ↪format(arr_uniform_rand_reshaped_var))
```

```
Means of the Array Column Wise: [0.5071 0.4074 0.3594 0.5023 0.4862]
Medians of the Array Column Wise: [0.5006 0.4438 0.3638 0.6271 0.5735]
Variances of the Array Column Wise: [0.1    0.054  0.0278 0.1127 0.0638]
```

Q6. Generate 50 random numbers from [0,1). Split this array into two equal arrays of size 25 each. Now reshape the two new arrays into 5x5 matrices and multiply both of them. Calculate the determinant of the new matrix (check wiki if you don't know what it means). Hint - you will find the determinant function part of the numpy linalg function (the linear algebra library)

Answer 6:

[15]:
```
# Generating a Uniform random Array

array_size = 50 # Enter a even number
np.random.seed(91) # Setting Seed as 91

arr_rand = np.random.rand(array_size)
```

[16]:
```
# Splitting & rehaping the Array into two parts

split_size = int(array_size/2)
newshape = (5,5)

arr_rand_part1 = arr_rand[0:split_size].reshape(newshape)
arr_rand_part2 = arr_rand[split_size:].reshape(newshape)

print('Array 1: \n\n {}'.format(arr_rand_part1))
print('\n\n Array 2: \n\n {}'.format(arr_rand_part2))
```

```
Array 1:

 [[0.201  0.329  0.2965 0.0933 0.3331]
 [0.7252 0.6524 0.5049 0.9473 0.6274]
 [0.9923 0.0761 0.3426 0.3069 0.7102]
 [0.9173 0.0809 0.0628 0.5822 0.6692]
 [0.0093 0.4219 0.3954 0.0732 0.7703]]


 Array 2:
```

```
[[0.1894 0.468  0.6517 0.0082 0.068 ]
 [0.7469 0.1575 0.5348 0.6763 0.5196]
 [0.5194 0.4658 0.2446 0.9115 0.1077]
 [0.4818 0.7487 0.385  0.7519 0.7596]
 [0.2879 0.6734 0.1756 0.672  0.2965]]
```

[17]:
```
# Matrix Multiplication of the two Arrays & Printing the Results

arr_multiplied = arr_rand_part1 @ arr_rand_part2
print('The multiplied array is: \n\n {}'.format(arr_multiplied))
```

```
The multiplied array is:

 [[0.5787 0.5782 0.4739 0.7884 0.3862]
 [1.5238 1.8091 1.4199 2.0412 1.3483]
 [0.7751 1.344  1.0141 1.0799 0.5876]
 [0.7399 1.3579 0.9981 1.0069 0.7518]
 [0.7793 0.8285 0.4919 1.2185 0.5464]]
```

[18]:
```
# Calculating the Determinant of the multiplied Array & Printing it

arr_multiplied_det = np.linalg.det(arr_multiplied)

print('The determinant of the multiplied array is: {:.8f}'.
 ↪format(arr_multiplied_det))
```

```
The determinant of the multiplied array is: 0.00009735
```

0.1 —
