تمرین اول هوش محاسباتی

بنفشه قلی نژاد – 98522328

- **سوال 1)**

با کمک کتاب خانه numpy ، می توان دو آرایه را عنصر به عنصر با عملگر های = , < , > مقایسه کرد و نتیجه به صورت یک nd-array و مقایسه آن ها به صورت True-False باز میگردد.

```
def element_wise_comparison(array1, array2):

    greater_result = array1 >array2
    greater_equal_result = array1 >= array2
    less_result = array1 < array2
    less_equal_result = array1 <= array2

    return greater_result, greater_equal_result, less_result,
less_equal_result
```

نتیجه ی کد بالا و با مثال های نوت بوک :

```
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[1, 2], [2, 3]])
greater, greater_equal, less, less_equal = element_wise_comparison(array1,
array2)
```

Greater than:

[[False False]

[True True]]

Greater than or equal to:

[[True True]

[True True]]

Less than:

[[False False]

[False False]]

Less than or equal to:

[[True True]

[False False]]

From: https://note.nkmk.me/en/python-numpy-ndarray-compare/#:~:text=In%20NumPy%2C%20you%20can%20compare,Functions%20such%20as%20np

- **سوال 2)**

با توجه به این که کتابخانه numpy ، برای هر کدام از این متد ها ، فانکشن مربوط به آن را پیاده سازی کرده است، Np.multiply دو ماتریس را عنصر به عنصر ضرب می کند و تابع np.matmul بین دو ماتریس ضرب ماتریسی را اعمال می کند بنابرین :

```
def array_multiply(array1, array2, method="element-wise"):

    if method == "element-wise":
      result = np.multiply(array1 , array2)
    elif method == "matrix-multiply":
      result = np.matmul(array1, array2)
    else:
     result = np.dot(array1 , array2)
    return result
```

نتیجه ی کد و با مثال های نوت بوک :

```
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[2, 0], [1, 2]])
```

Element-wise multiplication:

[[2 0]

[3 8]]

Matrix multiplication:

[[4 4]

[[8 10]

:From:https://www.educative.io/blog/numpy-matrix-multiplication

- **سوال 3)**

همان طور که مشخص است ، این سوال دو حالت row-wise و column-wise دارد. اول از همه باید چک کنیم که اندازه بردار ها طبق خواسته سوال باشد. بنابراین چک می کنیم که بردار اول n * n (برابر بودن سطر و ستون) و همچنین بردار دوم n*1باشد. در هر دو حالت به عملیات broadcasting نیاز داریم تا بتوانیم عناصر ماتریس دوم را به به ماتریس اول به صورت افقی یا عمودی اضافه کنیم.

Broadcasting ماتریس دوم را به اندازه ای که میخواهیم بسط می دهد

در حالت row-wise ، ماتریس دوم را به اندازه ی ماتریس اول بسط می دهیم تا از نظر اندازه یکی شوندو سپس با یک دیگر جمع می کنیم.

در حالت column-wiseو به کمک np.vstack ماتریس n*1 را به 1*n تبدیل می کنیم و مانند مرحله بالا به اندازه ماتریس اول broadcast می کنیم و سپس با هم جمع می کنیم.

```
def broadcast_add(p, q, method="row-wise"):
    if  p.shape[1] != p.shape[0] or p.shape[0] != q.shape[0] or
len(q.shape) > 1:
        raise ValueError(
            "shapes are incompatible for the chosen method")
    if method == "row-wise":
        result = p + np.broadcast_to(q, p.shape)
    elif method == "column-wise":
        result = p + np.broadcast_to(np.vstack(q), p.shape)

    return result
```

نتیجه :

```
# Example usage with different-shaped arrays
p = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
q = np.array([10, 20, 30])
```

Row-wise addition:

[[11 22 33]

[14 25 36]

[17 28 39]]

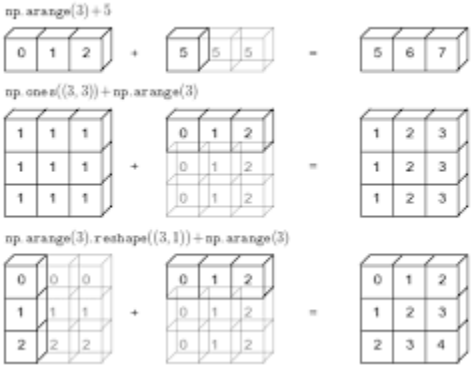
(3, 3) (3,)

Column-wise addition:

[[11 12 13]

[24 25 26]

[37 38 39]]

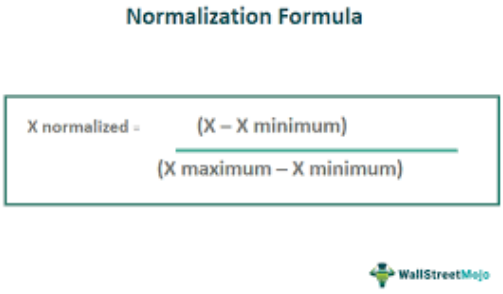


From: https://numpy.org/doc/stable/reference/generated/numpy.broadcast_to.html#numpy.broadcast_to

https://numpy.org/doc/stable/reference/generated/numpy.vstack.html

- **سوال 4)**

عملیات نرمالایریشن با کمک این فرمول انجام می پذیرد :



کد زده شده :

```
# Initialize the random matrix
x = np.random.randint(1, 11, (4, 4))
min_val = np.min(x)
max_val = np.max(x)
x = (x - min_val) / (max_val - min_val)
```

Original Array:

[[6 2 10 5]

 [4 3 3 10]

 [10 5 4 6]

 [7 2 4 2]]

After normalization:

[[0.5   0.   1.   0.375]

 [0.25 0.125 0.125 1.   ]

 [1.   0.375 0.25 0.5]

 [0.625 0.   0.25 0.   ]]

From: https://www.wallstreetmojo.com/normalization-formula/

- **سوال 5**

قسمت اول)برای محاسبه بازه ی روزانه، با کمک pd.Diff() اختلاف روز قبلی را با مقدار کنونی محاسبه می کند.
و shift(1)با شیفت دادن داده ها همه را به همان نسبت به مقدار قبلی آن تقسیم می کند. در نتیجه کد بازه برابر مقدار زیر است:

```
returns = data["Closing Price"].diff() / data["Closing Price"].shift(1)
```

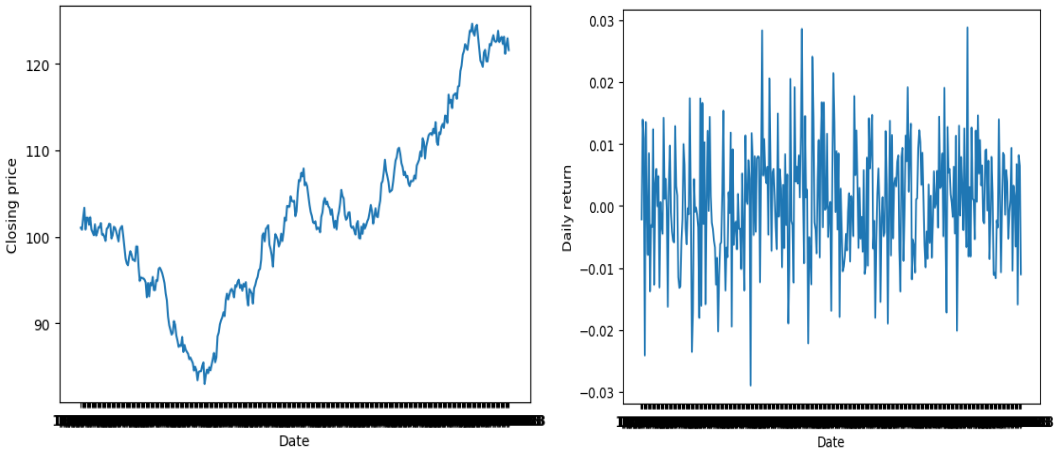
قسمت دوم و سوم) برای محاسبه ی میانگین و انحراف معیار می توانیم از np.meanبرای میانگین و np.std برای انحراف معیار پارامتر قبلی استفاده کنیم.

```
mean_return = np.mean(returns)
std_return = np.std(returns)
```

The mean daily return is 0.0005548260008486608

The standard deviation of the daily returns is 0.009442945103460247

قسمت چهارم و پنجم) برای نمایش دادن قیمت روزانه و میزان بازده از کتابخانه matplotlibاستفاده می کنیم. نتیجه به صورت زیر است:



قسمت ششم و هفتم)

برای پیداکردن بیشترین و کم ترین بازده از توابع max , min استفاده می کنیم.

```
max_price = data["Closing Price"].max()
min_price = data["Closing Price"].min()
```

همچنین برای تاریخ و بیشترین و کم ترین قیمت های تاریخی سهام از idxmaxاستفاده می کنیم.

```
max_price = data["Closing Price"].max()
min_price = data["Closing Price"].min()
max_price_date = data["Closing Price"].idxmax()
min_price_date = data["Closing Price"].idxmin()
print("The maximum daily return was", max_return, "on", max_return_date)
print("The minimum daily return was", min_return, "on", min_return_date)
print("The maximum price was", max_price, "on", max_price_date)
print("The minimum price was", min_price, "on", min_price_date)
```

نتیجه :

The maximum daily return was 0.02878633838810639 on 312

The minimum daily return was -0.028963574613605738 on 105

The maximum price was 124.6180108 on 332

The minimum price was 82.96821012 on 105

From : https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.diff.html

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shift.html

https://numpy.org/doc/stable/reference/generated/numpy.mean.html

https://numpy.org/doc/stable/reference/generated/numpy.std.html

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.idxmax.html

- **سوال 6)**

همان طور که در صورت سوال توضیح داده شده است، فرمول feedforwardبه صورت زیر است.

$$f = \sum_{i=1}^{n} x_i w_i$$

برای روش استفاده از حلقه، ابتدا خروجی را تعیین می کنیم که اندازه آن N*1 است. (به اندازه num_samples , 1) سپس همانند توضیح داده شده ، یک forتو در تو می زنیم و مجموع تمام حالت های ضرب ضریب در درایه های ماتریکس xرا برای هر المان output میریزم.

```
def for_loop_feed_forward(X, w):
    num_samples, num_features = X.shape
    outputs = np.zeros((num_samples, 1))

    for i in range(num_samples):
        for j in range(num_features):
            outputs[i, 0] += X[i, j] * w[j, 0]
    return outputs
```

برای روش vectorization، برای محاسبه این ماتریس می توانیم خیلی ساده از np.dot(x,w)استفاده کنیم و دقیقا همین عملیات بالا را برای ما پیاده سازی میکند.

$$\begin{bmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{bmatrix} . \begin{bmatrix} w1 \\ w2 \\ w3 \end{bmatrix} = \begin{bmatrix} f1 \\ f2 \\ f3 \end{bmatrix}$$

```
def vectorized_feed_forward(X, w):
    outputs = np.dot(X, w)
    return outputs
```

مقایسه خروجی و زمان صرف شده :

Time spent on calculating the outputs using for loops:

0.364208459854126

Time spent on calculating the outputs using vectorization:

0.004645586013793945

همان طور که مشخص است، زمان صرف شده برای حالت vectorization به صورت قابل توجهی کم تر است و می توان اهمیت این روش را در این مثال متوجه شد.

- **سوال 7)** برای این سوال می توانیم از تابع np.where استفاده کنیم. المان اول این تابع را شرطی که میخواهیم قرار میدهیم . (array> threshold) و دو مقادیر بعدی مقادیری که به ازای شرطمان میخواهیم باشد یعنی 1 و 0 .

```
def replace_elements_above_threshold(array, threshold):
    modified_arr = np.where(array > threshold, 1, 0)
    return modified_arr
```

مثال و نتیجه:

```
input_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
threshold_value = 5
```

[[0 0 0]

[0 0 1]

[1 1 1]]

From: https://numpy.org/doc/stable/reference/generated/numpy.where.html

- **سوال 8)**

همان طور که سوال خواسته و بخشی از کد زده شده است. یک کلاس داریم و در کانستراکتور آن ماتریسمان را که از جنس لیست است تعریف میکنیم .

```
class Matrix:
def __init__(self, my_matrix):
        self.matrix = my_matrix
```

*قسمت اول: برای این قسمت به طور خیلی ساده و باکمک یک فور تو در تو در صورتی که اندازه ماتریس ها برابر باشد ، مقایسه می کنیم و اگر تمام المان ها یکی بودند، Trueرا بر می گردانیم.

```
 def is_equal(self, second_matrix):
        m  = len(self.matrix)
        n = len(self.matrix[0])
        if len(self.matrix) != len(second_matrix.matrix) or
len(self.matrix[0]) != len(second_matrix.matrix[0]):
          return False
        else:
          for i in range(m):
            for j in range(n):
                if self.matrix[i][j] != second_matrix.matrix[i][j]:
                  return False
```

مثال :

```
matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = Matrix([[0, 0, 0], [4, 5, 6], [7, 8, 9]])
print(matrix1.is_equal(matrix2) , "matrix 1 equality")
```

False matrix 1 equality

*قسمت دوم: برای این قسمت نیز از یک فور تو در تو استفاده می کنیم و عناصر را یکی یکی مقایسه کرده و در صورت براورده شدن شرط True یا falseدر ماتریس خروجی قرار میدهیم.

```
  def is_higher_elementwise(self, second_matrix):
        output =  copy.deepcopy(self.matrix)
        m , n = len(self.matrix), len(self.matrix[0])
        for i in range(m):
          for j in range(n):
            if(self.matrix[i][j]  > second_matrix.matrix[i][j]):
              output[i][j] = True
            else:
              output[i][j] = False
        return Matrix(output)
```

مثال و نتیجه:

```
matrix3 = Matrix([[0, 0, 0], [10, 20, 30], [-1, 8, 10]])
```

[[True, True, True], [False, False, False], [True, False, False]]

*قسمت سوم: برای چک کردن این که یک ماتریس زیر مجموعه ماتریس دیگری است یا خیر ، از روش Naïve استفاده می کینم. به طوری که تمام زیر ماتریس های به اندازه زیر ماتریس گفته شده از ماتریس بزرگ تر را در نظر گرفته و در صورتی که یکی از آن زیر ماتریس ها با زیر ماتریس در نظر گرفته ما برابر باشد ، مقدار Trueریتزن می شود. در این کد activate برای زمانی است که یک زیر مجموعه ای پیدا کنیم که شبیه زیر مجموعه خواسته شده باشد. در صورتی که False باشد، حلقه تو درتوی دوم که به اندازه ماتریس کوچک تر و خواسته شده است ، شکسته می شود و به حلفه بزرگ تر برای پیدا کردن زیر ماتریس های دیگر می رود.

```
 def is_subset(self, second_matrix):
         m = len(self.matrix)
         n = len(self.matrix[0])
         activate = False
         if m > len(second_matrix.matrix) or n > len(second_matrix.matrix[0]):
          return False
        for k in range(len(second_matrix.matrix)):
          for v in range(len(second_matrix.matrix[0])):
            for i in range(m):
              for j in range(n):
                if(i + k <len(second_matrix.matrix) and j+v <
len(second_matrix.matrix[0])):
                  if self.matrix[i][j]  == second_matrix.matrix[i+k][j+v]:
                    activate = True
                  else:
                    activate =False
              if activate == False :
                break;
            if activate == True:
            return True
        return False
```

نتیجه و مقایسه با ماتریس 1)

```
matrix4 = Matrix([[5, 6], [8, 9]]) #True
matrix5 = Matrix([[1, 2], [4,5]]) #True
matrix6 = Matrix([[1, 2], [3, 4]]) #False
```

نتیجه)

True matrix4

True matrix5

False matrix6

*قسمت 8) : الگوریتم ضرب ماتریس را برای این قسمت پیاده می کینم. ابتدا باید چک کنیم که تعداد ستون های ماتریس اول با تعداد سطر های ماتریس دوم برابر باشد(شرط ضرب ماتریس)

اندازه ماتریس خروجی به اندازه سطر های ماتریس اول و ستون های ماتریس دوم است

سپس با کمک یک فور و یک فور سوم (برای عملیات ضرب درایه ها و جمعشان با یک دیگر) عملیات ضرب را پیاده می کنیم و در نهایت مشاهده می کنیم که نتیجه np.dot با تابع پیاده سازی شده یکسان است.

```
  def dot_product(self, second_matrix):
        if len(self.matrix[0]) != len(second_matrix.matrix):
          raise ValueError("The two matrices must have the same number of
columns.")
        result = [[0 for i in range(len(second_matrix.matrix[0]))] for j
in range(len(self.matrix))]
        for i in range(len(self.matrix)):
          for j in range(len(second_matrix.matrix[0])):
            for k in range(len(self.matrix[0])):
              result[i][j] += self.matrix[i][k] *
second_matrix.matrix[k][j]
        return Matrix(result)
```

مثال و نتیجه :

```
matrix7 = Matrix([[3, 1], [2, 4], [-1, 5]])
matrix8 = Matrix([[3, 1], [2, 4]])
print( "matrix result", matrix7.dot_product(matrix8).matrix)
print( "numpy result" , np.dot(matrix7.matrix, matrix8.matrix))
```

matrix result :[[11, 7], [14, 18], [7, 19]]

numpy result: [[11 7]

 [14 18]

 [7 19]]

همان طور که مشخص است نتیجه یکسان است.