به نام خالق رنگین کمان

ستاره باباجانی- گزارش تمرین اول

سوال1: در این سوال از ما خواسته شد با استفاده از کتابخانه numpy تابعی برای مقایسه عنصر به عنصر دو آرایه طراحی کنیم. در شکل زیر مشاهده میشود که از توابع greater, greater_equal, less, less_equal برای این منظور استفاده شده است.

```python
def element_wise_comparisons(array1, array2):
    """
    Perform element-wise comparisons between two NumPy arrays.

    Parameters:
    - array1 (numpy.ndarray): First input NumPy array.
    - array2 (numpy.ndarray): Second input NumPy array.

    Returns:
    - tuple: A tuple of NumPy arrays containing the following element-wise comparison results:
        - greater_result (numpy.ndarray): Element-wise greater than comparison.
        - greater_equal_result (numpy.ndarray): Element-wise greater than or equal to comparison.
        - less_result (numpy.ndarray): Element-wise less than comparison.
        - less_equal_result (numpy.ndarray): Element-wise less than or equal to comparison.
    """
    greater_result = np. greater(array1, array2)
    greater_equal_result = np.greater_equal(array1, array2)
    less_result = np.less(array1, array2)
    less_equal_result = np.less_equal(array1, array2)

    return greater_result, greater_equal_result, less_result, less_equal_result
```

تست و نتیجه آن، در عکس های زیر قابل مشاهده است و همان طور که مشاهده میشود عملکرد تابع درست است:

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

greater, greater_equal, less, less_equal = element_wise_comparisons(array1, array2)

print("Greater than:")
print(greater)
print("\nGreater than or equal to:")
print(greater_equal)
print("\nLess than:")
print(less)
print("\nLess than or equal to:")
print(less_equal)
```

```
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]]
```

سوال2: در این سوال، طبق متد ورودی یکی از دو روش element-wise و یا matrix_multiply برای ضرب دو ماتریس انتخاب میشود. در روش اول بصورت عنصر به عنصر این ضرب با استفاده از تابع multiply از کتابخانه numpy انجام میشود و در روش دوم با استفاده از تابع matmul ضرب عادی دو ماتریس انجام میشود.

```python
def array_multiply(array1, array2, method="element-wise"):
    """
    Perform multiplication between two NumPy arrays using the specified method.

    Parameters:
    - array1 (numpy.ndarray): First input NumPy array.
    - array2 (numpy.ndarray): Second input NumPy array.
    - method (str, optional): The multiplication method to use. Defaults to "element-wise".

    Returns:
    - numpy.ndarray: The result of the multiplication operation based on the chosen method.
    """
    if method == "element-wise":
      result = np.multiply(array1, array2)
    else:
      result = np.matmul(array1, array2)

    return result
```

حال تست و نتیجه آن در عکس های زیر قابل مشاهده است:

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

# Perform element-wise multiplication
element_wise_result = array_multiply(array1, array2, method="element-wise")
print("Element-wise multiplication:")
print(element_wise_result)

# Perform matrix multiplication
matrix_multiply_result = array_multiply(array1, array2, method="matrix-multiply")
print("\nMatrix multiplication:")
print(matrix_multiply_result)
```

```
Element-wise multiplication:
[[2 0]
 [3 8]]

Matrix multiplication:
[[ 4  4]
 [10  8]]
```

سوال3: در این سوال طبق متد ورودی، دو آرایه بصورت افقی یا بصورت عمودی بهم اضافه میشوند. برای حالت اول، ابتدا چک میشود که دو آرایه row های یکسان داشته باشند و در غیر اینصورت، ارور برگردانده میشود. اگر row ها یکسان بود، دو آرایه به دستور جمع عادی، جمع میشوند.(اگر تعداد ستون ها یکی نبود، broadcast میشود، پس مشکلی نیست!). در روش دوم، باز هم اگر row ها تفاوتی نداشتند، این بار آرایه q بصورت عمودی با آرایه p جمع میشود و اگر تعداد سطر ها یکی نبود، broadcast میشود.

```python
def broadcast_add(p, q, method="row-wise"):
    """
    Perform addition between two NumPy arrays using broadcasting and the specified method.

    Parameters:
    - p (numpy.ndarray): First input NumPy array.
    - q (numpy.ndarray): Second input NumPy array.
    - method (str, optional): The addition method to use. Defaults to "row-wise".
        - "row-wise": Perform row-wise addition, broadcasting q to match the number of rows in p.
        - "column-wise": Perform column-wise addition, adding q to each column of p.

    Returns:
    - numpy.ndarray: The result of the addition operation based on the chosen method.

    Raises:
    - ValueError: If an invalid method is provided or if the shapes are incompatible for the chosen method.
    """
    if method == "row-wise":
        if p.shape[0] != q.shape[0]:
            raise ValueError("Incompatible shapes for row-wise addition. Number of rows must match.")
        result = p + q
    elif method == "column-wise":
        if p.shape[0] != q.shape[0]:
            raise ValueError("Incompatible shapes for column-wise addition. Number of columns must match.")
        result = p + q[:, np.newaxis]
    else:
        raise ValueError("Invalid method. Choose either 'row-wise' or 'column-wise'.")

    return result
```

تست و نتیجه آن در عکس های زیر قابل مشاهده هستند:

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

# Add q row-wise to p
row_wise_result = broadcast_add(p, q, method="row-wise")
print("Row-wise addition:")
print(row_wise_result)

# Add q column-wise to p
column_wise_result = broadcast_add(p, q, method="column-wise")
print("\nColumn-wise addition:")
print(column_wise_result)
```

```
Row-wise addition:
[[11 22 33]
 [14 25 36]
 [17 28 39]]

Column-wise addition:
[[11 12 13]
 [24 25 26]
 [37 38 39]]
```

سوال4: در این سوال ابتدا با دستور random.randint یک ماتریس با عناصر رندوم تشکیل شد و سپس بر اساس فرمول ذکر شده در کد(با توجه به مقدار ماکسیمم و مقدار مینیمم عناصر آن ماتریس)، فرآیند نرمالایز کردن انجام شد.

```python
# Initialize the random matrix
x = np.random.randint(1, 10, size=(4, 4))

print("Original Array:")
print(x)

# Do the normalization
# based on that for this kind of normalization we should decrease the min value of x from it and then divide to (max value of x - min value of it)
min_value = x.min()
max_value = x.max()
x = (x - min_value) / (max_value - min_value)

print("After normalization:")
print(x)
```

```
Original Array:
[[3 5 3 4]
 [8 1 7 6]
 [1 1 9 1]
 [9 6 4 8]]
After normalization:
[[0.25  0.5   0.25  0.375]
 [0.875 0.    0.75  0.625]
 [0.    0.    1.    0.   ]
 [1.    0.625 0.375 0.875]]
```

سوال5: در این سوال ابتدا فایل دیتای داده شده خوانده شد. حال به کد و خروجی آن برای هر بخش میپردازیم:

- میزان بازده روزانه: طبق فرمول داده شده، مقدار بازده روزانه محاسبه شد و در ستون جدیدی در فایل دیتا، قرار گرفت(چون بعدا از آن استفاده میشود):

```python
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv("data.csv")
#.shift(1) = Closing Price rouz ghabl
data['daily profit'] = (data['Closing Price'] - data['Closing Price'].shift(1)) / data['Closing Price'].shift(1)
daily_returns = data['daily profit']
print(daily_returns)
print("\n")
```

```
0           NaN
1     -0.002145
2      0.013911
3      0.010884
4     -0.024109
         ...
359    0.006720
360   -0.015886
361    0.008170
362    0.006454
363   -0.011010
Name: daily profit, Length: 364, dtype: float64
```

همانطور که نوشته شده است، دستور shift برای گرفتن دیتای روز قبل زده شده است.

- میانگین بازده روزانه: این مقدار با استفاده از تابع mean محاسبه شد.

```
10
11 average_daily_return = daily_returns.mean()
12 print(f'average_daily_return: {average_daily_return}')
13 print("\n")
14
```

```
average_daily_return: 0.0005548260008486608
```

- انحراف معیار بازده روزانه: این مقدار با استفاده از تابع std محاسبه شد.

```
15 std_dev_daily_return = daily_returns.std()
16 print(f'std_dev_daily_return: {std_dev_daily_return}')
17 print("\n")
18
```

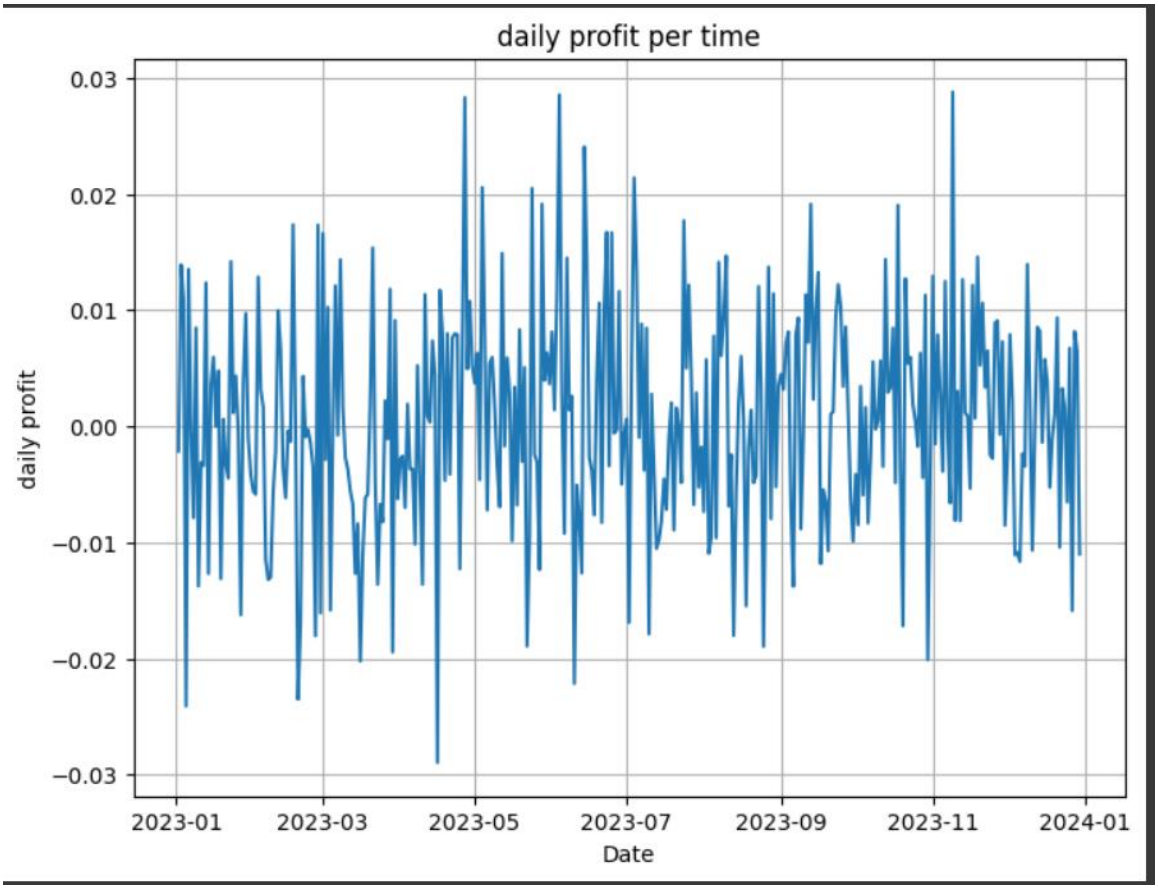
```
std_dev_daily_return: 0.009455978850317194
```

- قیمت های پایانی روزانه سهام: با استفاده از matplotlib نموداری شرح داده شد که محور افقی آن روزها و محور عمودی آن قیمت های پایانی آن روز است.

```
19 data['Date'] = pd.to_datetime(data['Date'])
20 plt.figure(figsize=(8, 6))
21 plt.plot(data['Date'], data['Closing Price'])
22 plt.xlabel('Date')
23 plt.ylabel('daily closing price')
24 plt.title('daily closing price per time')
25 plt.grid(True)
26 plt.show()
27 print("\n")
28
```

- میزان بازده روزانه به مرور زمان: محور افقی این نمودار روز(تاریخ) و محور عمودی آن مقدار بازده آن روز که در ستون جدیدی ذخیره شده بود، است.

```
29 plt.figure(figsize=(8, 6))
30 plt.plot(data['Date'], data['daily profit'])
31 plt.xlabel('Date')
32 plt.ylabel('daily profit')
33 plt.title('daily profit per time')
34 plt.grid(True)
35 plt.show()
36 print("\n")
37
```



- روز هایی با بیشترین و کمترین بازده: این روزها از طریق idxmax, idxmin در ستون جدید اضافه شده، پیدا شده و سپس مقدار بازده آنها پرینت شده است.

```
38 max_return_day = daily_returns.idxmax()
39 min_return_day = daily_returns.idxmin()
40 print(f'max_return_day: {data.at[max_return_day, "Date"]} with return of {daily_returns[max_return_day]}')
41 print(f'min_return_day: {data.at[min_return_day, "Date"]} with return of {daily_returns[min_return_day]}')
42 print("\n")
43
```

```
max_return_day: 2023-11-09 00:00:00 with return of 0.028786338838810639
min_return_day: 2023-04-16 00:00:00 with return of -0.028963574613605738
```

- تاریخ و مقدار بیشترین و کمترین قیمت های تاریخی سهام: این تاریخ از طریق idxmax, idxmin در ستون closing price پیدا شده و سپس مقدار آن پرینت شده است.

```python
max_price_date = data.loc[data['Closing Price'].idxmax(), 'Date']
max_price_value = data['Closing Price'].max()
min_price_date = data.loc[data['Closing Price'].idxmin(), 'Date']
min_price_value = data['Closing Price'].min()
print(f'max_price_date: {max_price_date} with price of {max_price_value}')
print(f'min_price_date: {min_price_date} with price of {min_price_value}')
# You should write your code here and print or plot the required data asked in homework documentation
```

```
max_price_date: 2023-11-29 00:00:00 with price of 124.6180108
min_price_date: 2023-04-16 00:00:00 with price of 82.96821012
```

سوال6: در این سوال feed forward به دو روش حلقه(for) عادی و vectorization(با numpy) زده شد و همان طور که در نتیجه سرعت آن مشاهده میشود، روش دوم به مراتب سریعتر است و مدت زمان کمتری را برای اجرا صرف میکند.

```python
def vectorized_feed_forward(X, w):
    """
    Perform a feed-forward operation using vectorization.

    Parameters:
    - X (numpy.ndarray): Input data matrix of shape (num_samples, num_features).
    - w (numpy.ndarray): Weight matrix of shape (num_features, 1).

    Returns:
    - numpy.ndarray: Output matrix of shape (num_samples, 1).
    """
    outputs = np.dot(X, w)

    return outputs
```

```python
def for_loop_feed_forward(X, w):
    """
    Perform a feed-forward operation using a for loop.

    Parameters:
    - X (numpy.ndarray): Input data matrix of shape (num_samples, num_features).
    - w (numpy.ndarray): Weight matrix of shape (num_features, 1).

    Returns:
    - numpy.ndarray: Output matrix of shape (num_samples, 1).
    """
    outputs = np.zeros((1000, 1))
    for i in range(1000):
        outputs[i, 0] = np.sum(X[i, :] * w)
    return outputs

```

خروجی و نتیجه مقایسه سرعت این دو روش در عکس های زیر قابل مشاهده است:

```python
import time

# generate random samples

X = np.random.rand(1000, 500)
w = np.random.rand(500, 1)

start_time = time.time()
outputs = for_loop_feed_forward(X, w)

print("Time spent on calculating the outputs using for loops: ")
print(time.time() - start_time)

start_time = time.time()
outputs = vectorized_feed_forward(X, w)

print("Time spent on calculating the outputs using vectorization: ")
print(time.time() - start_time)
```

```
Time spent on calculating the outputs using for loops:
0.23777079582214355
Time spent on calculating the outputs using vectorization:
0.004640340805053711
```

سوال7: در این سوال تابعی نوشته شد که طبق مقدار threshold ورودی، عناصر آرایه خود را تغییر میدهد.(اگر بیشتر از این مقدار باشند، یک و در غیر اینصورت مقدار ایندکس آن صفر میشود.)

```python
def replace_elements_above_threshold(array, threshold):
    """
    Replace elements in a NumPy array that are higher than the given threshold with a specified value.

    Parameters:
    - array (numpy.ndarray): Input NumPy array.
    - threshold (float): Threshold value to compare elements with.

    Returns:
    - numpy.ndarray: NumPy array with elements replaced above the threshold.
    """
    modified_arr = np.where(array > threshold, 1, 0)
    return modified_arr
```

تست و نتیجه آن در عکس زیر قابل مشاهده است و همان طور که مشاهده میشود، عملکرد تابع درست است:

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

```
[[0 0 0]
 [0 0 1]
 [1 1 1]]
```

سوال8: در این سوال کلاسی برای ساختار ماتریس تعریف شده است.

```
class Matrix:
    def __init__(self, matrix):
        """
        Initialize a Matrix object with a given list of lists.

        Parameters:
        - matrix (list of lists): Input list of lists representing the matrix.
        """
        self.matrix = matrix

```

کد هر متد و تست آن در عکس های زیر، قابل مشاهده هستند:

- متد مقایسه دو ماتریس برای تشخیص تساوی بودن آنها:

```
    def is_equal(self, second_matrix):
        """
        Check if this Matrix object is equal to another Matrix object.

        Parameters:
        - second_matrix (Matrix): Another Matrix object for comparison.

        Returns:
        - bool: True if the matrices are equal, False otherwise.
        """
        return self.matrix == second_matrix
```

```
matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = Matrix([[0, 0, 0], [4, 5, 6], [7, 8, 9]])

# test equality of matrices here and show the result #
is_equal = matrix1.is_equal(matrix2)
print(is_equal)

```

```
False
```

- متد مقایسه المان های دو ماتریس: ابتدا یک ماتریس مثل ماتریس اولیه ساخته شد و سپس با مقایسه المان به المان دو ماتریس، درصورت بزرگتر بودن مقدار المان ماتریس اول، مقدار True در ماتریس نتیجه ذخیره میشود و در غیر اینصورت، مقدار False قرار داده میشود:

```
    def is_higher_elementwise(self, second_matrix):
        """
        Check if this Matrix object has higher values element-wise compared to another Matrix object.

        Parameters:
        - second_matrix (Matrix): Another Matrix object for comparison.

        Returns:
        - Matrix: Matrix same shape of the input.
        """
        result = Matrix([[False for i in range(len(self.matrix))] for j in range(len(self.matrix[0]))])
        for i in range(len(self.matrix)):
            for j in range(len(self.matrix[i])):
                if (self.matrix[i][j] > second_matrix.matrix[i][j]):
                    result.matrix[i][j] = True
        return result.matrix
```

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

# test proportion of matrices here and show the result #
comparison_result = matrix3.is_higher_elementwise(matrix2)
print(comparison_result)


```

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

- متد برای بررسی زیرمجموعه بودن دو ماتریس: برای این منظور، همان طور که در کد نوشته شده است، ابتدا تمام زیرمجموعه های ماتریس اولیه محاسبه شده و سپس از بین این آرایه تشکیل شده، اگر یکی برابر با ماتریس دوم بوده باشد، مقدار True بازگردانده میشود و در غیر اینصورت یعنی ماتریس دوم زیرمجموعه نیست و False بازگردانده میشود:

```
    def is_subset(self, second_matrix):
        """
        Check if this Matrix object is a subset of another Matrix object.

        Parameters:
        - second_matrix (Matrix): Another Matrix object for comparison.

        Returns:
        - bool: True if this matrix is a subset of 'second_matrix', False otherwise.
        """
        #in order to define the subset of a matrix, we prepare all its subsets and then compare them with our second one. if one of them is equal to it, then we return True!
        subsets = []
        subset = [[0 for _ in range(len(second_matrix.matrix[0]))] for _ in range(len(second_matrix.matrix))]
        for i in range(len(self.matrix) - len(second_matrix.matrix) + 1):
            for j in range(len(self.matrix[0]) - len(second_matrix.matrix[0]) + 1):
                subset = [row[j : j + len(second_matrix.matrix[0])] for row in self.matrix[i : i + len(second_matrix.matrix)]]
                subsets.append(subset)

        for subset in subsets: #checking the subsets
            if second_matrix.is_equal(Matrix(subset)):
                return True
        return False
```

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


# test subset of matrices here and show the result #
print(matrix1.is_subset(matrix6))


```

```
False
```

- متد برای ضرب دو ماتریس: برای این متد ماتریسی مثل ماتریس ورودی ساخته شده و طبق خاصیت ضرب ماتریس ها و با استفاده از سه حلقه تو در تو، این ضرب محاسبه میشود:

```python
def dot_product(self, second_matrix):
    """
    Calculate the dot product between this Matrix object and another Matrix object.

    Parameters:
    - second_matrix (Matrix): Another Matrix object for the dot product.

    Returns:
    - Matrix: The result of the dot product as a numpy.ndarray.
    """
    result = [[0 for _ in range(len(self.matrix[0]))] for _ 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 result
```

```python
matrix7 = Matrix([[3, 1], [2, 4], [-1, 5]])
matrix8 = Matrix([[3, 1], [2, 4]])

# test product of matrices here and show the result #
result_multiply = matrix7.dot_product(matrix8)
print(result_multiply)


```

```
[[11, 7], [14, 18], [7, 19]]
```

پایان گزارش