گزارش تمرین CI_HW1 – محمد اصولیان 99521073

Q1)

با توجه به این که جنس ورودی‌ها از جنس آرایه numpy هستند، از عملگرهای بزرگتری کوچکتری خود numpy استفاده کردم:

```
# ------------
greater_result = array1 > array2
greater_equal_result = array1 >= array2
less_result = array1 < array2
less_equal_result = array1 <= array2
# ------------
```

نتیجه تست برای آرایه‌های نمونه مشخص شده:

```
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]]
```

Q2)

با چک کردن method، ضرب‌های خواسته شده را به کمک توابع numpy انجام دادم:

```
# ------------
result = np.multiply(array1, array2) if method == 'element-wise' else np.dot(array1, array2)
# ------------
```

چون در صورت سوال مشخص نشده بود که مقدار متغیر method برای حالت دیگر یعنی ضرب ماتریسی چه هست، به ازای هر مقداری جز element-wise، ضرب ماتریسی انجام دادم.

نتیجه تست:

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

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

Q3)

ابتدا مقدار method را چک کردم و در صورتی که متد پشتیبانی نمی‌شد، ValueError دادم. سپس با توجه به مقدار method، عملیات خواسته شده را انجام دادم

```python
# ------------
if method != "row-wise" and method != 'column-wise':
    raise ValueError('method is not supported')
try:
    if method == 'row-wise':
        q = q.reshape(1, p.shape[1])
        result = p+q
    if method == 'column-wise':
        q = q.reshape(p.shape[0], 1)
        result = p+q
except Exception:
    raise ValueError('shapes are incompatible')
# ------------
```

برای این کار از توابع broadcasting خود numpy استفاده کردم. در صورتی که عملیات row-wise باشد، ماتریس دوم را به حالت سطری و در غیر این صورت به حالت ستونی درآوردم تا numpy اروری ندهد. در صورتی که ابعاد این دو ماتریس با هم تطابق نداشته باشند، ValueError داده می‌شود.

نتیجه تست:

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

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

Q4

ماتریس اولیه را با استفاده از np.random.randint ایجاد کردم

```
# Initialize the random matrix
x = np.random.randint(1, 11, (4, 4))
```

سپس با توجه به مقدار مینیمم و ماکسیسمم، نرمالایزیشن انجام دادم.

```
# Do the normalization
x = x - x.min()
x = x / x.max()
```

نتیجه تست:

```
Original Array:
[[10  7  3  7]
 [ 7  9  4  8]
 [10  3  1  3]
 [ 2  3  8  5]]
After normalization:
[[1.         0.66666667 0.22222222 0.66666667]
 [0.66666667 0.88888889 0.33333333 0.77777778]
 [1.         0.22222222 0.         0.22222222]
 [0.11111111 0.22222222 0.77777778 0.44444444]]
```

Q5

فایل را به کمک pd.read_csv خواندم و مقادیر closing price را در یک آرایه np ذخیره کردم:

```
# You should write your code here and print
df = pd.read_csv('data.csv')
cp = df['Closing Price'].to_numpy()
```

Q5-1

طبق فرمول گفته شده مقدار بازده روزانه را حساب کردم و ده مقدار اول را برای تست نشان دادم.

```
# query1
query1 = (cp[1:] - cp[0:-1]) / cp[0:-1]
print(f'query1:\nDaily return for first 10 days starting from day 2: {query1[:10]}')
```

نتیجه:

```
query1:
Daily return for first 10 days starting from day 2: [-0.00214496  0.0139108   0.010
 -0.0078813   0.00847409 -0.01376749 -0.00311912]
```

(Q5-2 , Q5-3

برای محاسبه میانگین و انحراف معیار از توابع np.mean و np.std استفاده کردم

```python
# query2
query2 = query1.mean()
print(f'query2:\naverage daily return = {query2}')

# query3
query3 = query1.std()
print(f'query3:\nstandard deviation = {query3}')
```

نتیجه:

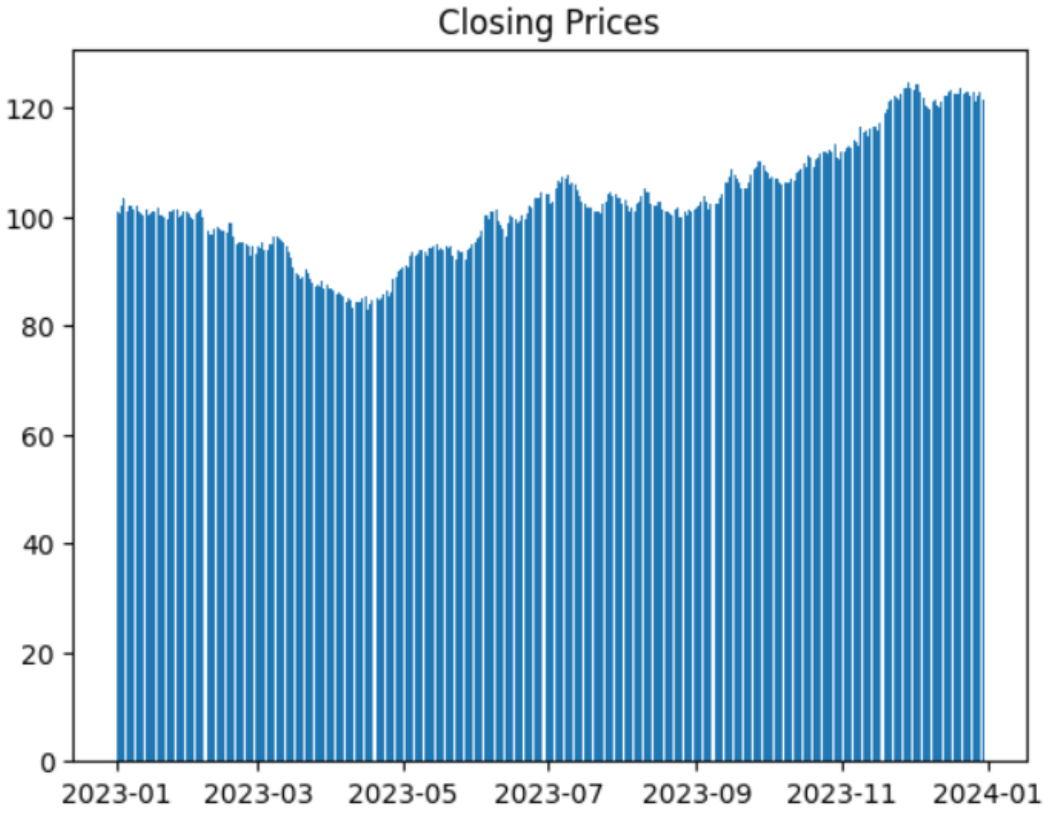
```
query2:
average daily return = 0.0005548260008486608
query3:
standard deviation = 0.009442945103460247
```

(Q5-4

ابتدا تاریخ هایی که به صورت string بودند رو به کمک datetime.strptime به فرمت تاریخ درآوردم. بعد به کمک plt.bar و مقادیر closing price که از قبل ذخیره داشتم، نمودار را رسم کردم

```python
# query4
dates = df['Date']
plt.title('Closing Prices')
plt.bar([datetime.datetime.strptime(date, r'%m/%d/%Y') for date in dates], cp)
plt.show()
```

نتیجه:

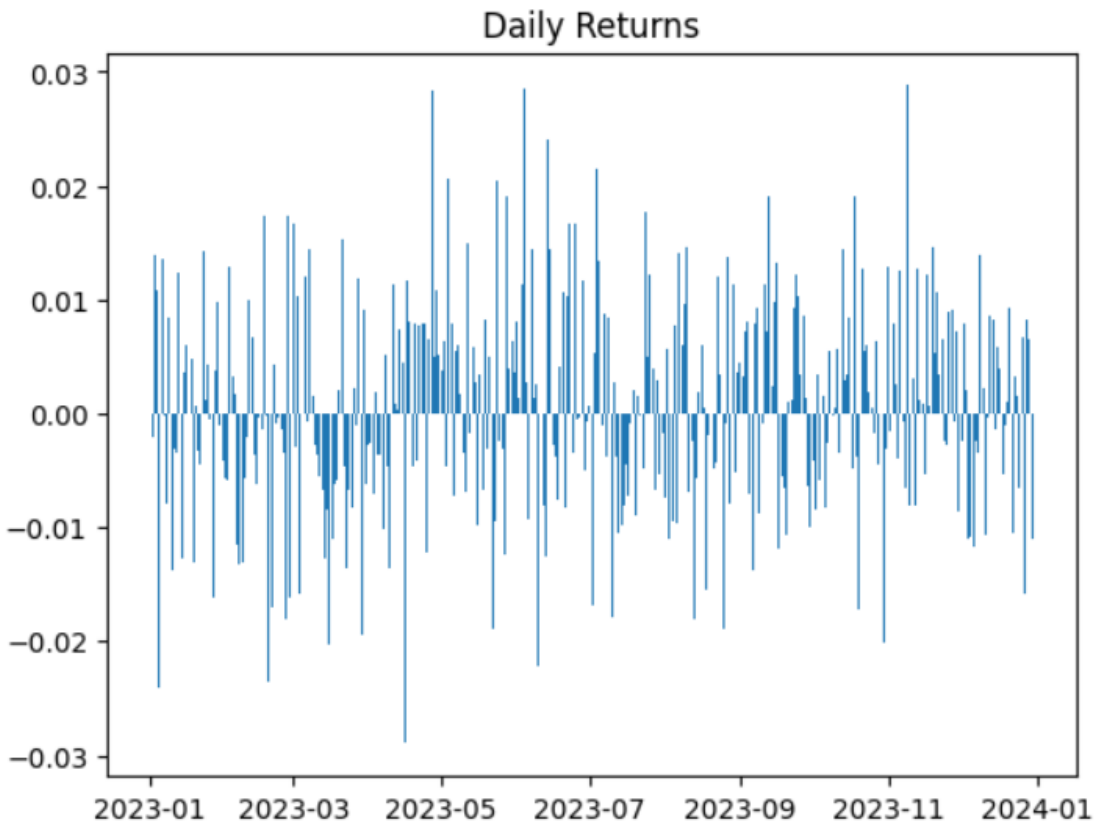


Q5-5)

مثل سوال قبل، فقط به جای مقادیر closing price از مقادیر بازده روزانه که در بخش اول به دست آورده بودم استفاده کردم

```
# query5
plt.title('Daily Returns')
plt.bar([datetime.datetime.strptime(date, r'%m/%d/%Y') for date in dates[1:]], query1)
plt.show()
```

نتیجه:



Q5-6)

بیشترین مقدار و اندیس آن را به کمک np.max و np.argmax پیدا کردم و به کمک اندیس، روز مربوطه رو پیدا کردم. برای کم ترین مقدار هم همینطور

```
# query6
best_day = dates[query1.argmax()+1]
worst_day = dates[query1.argmin()+1]
print(f'query6:\n{best_day=} return ={query1.max()}\n{worst_day=} return = {query1.min()}')
```

نتیجه:

```
query6:
best_day='11/9/2023' return =0.02878633838810639
worst_day='4/16/2023' return = -0.028963574613605738
```

Q5-7)

باز مانند سوال قبل عمل کردم اما با مقادیر closing price

```
# query7
best_price_date = dates[cp.argmax()]
worst_price_date = dates[cp.argmin()]
print(f'query7:\n{best_price_date=} price = {cp.max()}\n{worst_price_date=} price = {cp.min()}')
```

نتیجه:

```
query7:
best_price_date='11/29/2023' price = 124.6180108
worst_price_date='4/16/2023' price = 82.96821012
```

Q6)

برای قسمت حلقه، یک حلقه روی نمونه ها زدم و برای هر نمونه یک حلقه زدم و مقادیر فیچر و وزن را ضرب و جمع کردم و به این صورت آرایه f را پر کردم

```
# ------------
outputs = np.zeros((X.shape[0], 1))
for i in range(0, X.shape[0]):
    s = 0
    for j in range(0, X.shape[1]):
        s += X[i][j] * w[j]
    outputs[i] = s
# ------------
```

اما برای قسمت vectorization، تنها به کمک تابع np.dot، ماتریس X و w را در هم ضرب کردم

```
# ------------
outputs = np.dot(X, w)
# ------------
```

نتیجه اجرا ها به این صورت بود:

```
Time spent on calculating the outputs using for loops:
1.7349886894226074
Time spent on calculating the outputs using vectorization:
0.00099945068359375
```

میبینیم که در اجرا به کمک vectorization، عملیات تا هزار برابر سریع تر انجام می‌شود. این به خاطر این است که numpy، این عملیات ها را به کمک کدهای c++ که سرعت بیشتری دارند اجرا میکند و نکته مهمتر این است که ضرب ماتریسی را به صورت parallel انجام میدهد.

Q7

از سینتکس مخصوص آرایه‌های numpy برای انتخاب عناصر آرایه که از threshold بیشتر هستند استفاده کردم و مقدار آنها را با 1 جایگزین کردم. سایر عناصر هم به همین ترتیب با 0 جایگزین کردم

```
# ------------
modified_arr = array.copy()
modified_arr[array > threshold] = 1
modified_arr[array <= threshold] = 0
# ------------
```

نتیجه:

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

Q8

Q8-1

ابتدا ابعاد ماتریس ها را چک کردم و در صورتی که برابر نبودند، مقدارFalse خروجی دادم. سپس روی تمام اعضای ماتریس‌ها حلقه زدم و در صورتی که یک عنصر نابرابر داشتند، False خروجی دادم

```
# ------------
if not (len(self.values) == len(second_matrix.values) and len(self.values[0]) == len(second_matrix.values[0])):
    return False
equal = True
for i in range(len(self.values)):
    for j in range(len(self.values[0])):
        if self.values[i][j] != second_matrix.values[i][j]:
            equal = False
            break
    if not equal:
        break
return equal
# ------------
```

تست را به این صورت نوشتم و با موفقیت اجرا شد:

```
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 #
assert matrix1.is_equal(matrix2) == False
assert matrix1.is_equal(matrix1) == True
assert matrix2.is_equal(matrix2) == True
print('Matrix.is_equal works fine!')
```

Q8-2)

با استفاده از generator های لیست در پایتون، عناصر را با هم مقایسه کرم و مقدار False یا True را برای هر خانه انتخاب کردم

```
# ------------
result = [[True if self.values[i][j] > second_matrix.values[i][j] else False \
          for j in range(len(self.values[0]))] for i in range(len(self.values))]
return result
# ------------
```

نتیجه تست:

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

# test proportion of matrices here and show the result #
assert matrix1.is_higher_elementwise(matrix2) == [[True, True, True], [False, False, False], [False, False, False]]
assert matrix1.is_higher_elementwise(matrix3) == [[True, True, True], [False, False, False], [True, False, False]]
assert matrix2.is_higher_elementwise(matrix3) == [[False, False, False], [False, False, False], [True, False, False]]
print('Matrix.is_higher_elementwise works fine!')
```

✓ 0.0s

```
Matrix.is_higher_elementwise works fine!
```

Q8-3)

ابتدا چک کردم که ابعاد ماتریس زیرمجموعه، از ابعاد ماتریس دیگر کمتر باشد. سپس تمام حالات ممکن وجود یک ماتریس به ابعاد ماتریس زیرمجموعه را در ماتریس بزرگ تر بررسی کردم و با استفاده از تابع is_equal که عقب تر نوشته بودم، تساوی این حالات با ماتریس زیرمجموعه را بررسی کردم. در صورتی که هیچ حالتی صدق نکرد، False برگرداندم.

```
# ------------
result = [[0 for j in range(len(second_matrix.values[0]))] for i in range(len(self.values))]

if len(self.values[0]) != len(second_matrix.values):
    raise ValueError('matrix shapes are incompatible')
for i in range(len(self.values)):
    for j in range(len(second_matrix.values[0])):
        row = self.values[i]
        column = [row[j] for row in second_matrix.values]
        s = 0
        for k in range(len(row)):
            s += row[k] * column[k]
        result[i][j] = s
return result
# ------------
```

نتیجه تست:

```
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 #
assert matrix4.is_subset(matrix1) == True
assert matrix5.is_subset(matrix1) == True
assert matrix6.is_subset(matrix1) == False
print('Matrix.is_subset works fine!')
```

✓ 0.0s

```
Matrix.is_subset works fine!
```

Q8-4)

ابتدا ابعاد ماتریس اول و دوم را چک کردم و در صورتی که با هم تطابق نداشتند، ValueError دادم. سپس روی سطرهای ماتریس اول و ستون های ماتریس دوم حلقه زدم و عناصر را در هم ضرب و جمع کردم و ماتریس نتیجه را خانه به خانه پر کردم

```
# ------------
result = [[0 for j in range(len(second_matrix.values[0]))] for i in range(len(self.values))]

if len(self.values[0]) != len(second_matrix.values):
    raise ValueError('matrix shapes are incompatible')
for i in range(len(self.values)):
    for j in range(len(second_matrix.values[0])):
        row = self.values[i]
        column = [row[j] for row in second_matrix.values]
        s = 0
        for k in range(len(row)):
            s += row[k] * column[k]
        result[i][j] = s
return result
# ------------
```

نتیجه تست:

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

# test product of matrices here and show the result #
assert matrix7.dot_product(matrix8) == [[11, 7], [14, 18], [7, 19]]
print('Matrix.dot_product works fine!')
```

.5] ✓ 0.0s

```
Matrix.dot_product works fine!
```