تمرین اول
ملیکا محمدی فخار- ۹۹۵۲۲۰۸۶

۱. در این سوال برای مقایسه ماتریس‌های داده شده از توابع آماده‌ی numpy استفاده شده است که درایه‌های نظیر ماتریس‌های ورودی را با یکدیگر مقایسه می‌کند و یک بولین True, False برمی‌گرداند. نهایتا خروجی تابع به صورت یک تاپل بازگردانده شده است.

۲. در این سوال بر روی دو ماتریس ورودی، بسته به متد داده شده نوعی ضرب اجرا می‌شود. به این صورت که اگر متد element-wise بود، درایه‌های نظیر ماتریکس‌ها در هم ضرب می‌شوند و اگر matrix-multiply بود، ضرب ماتریسی (dot-product) صورت می‌گیرد.

۳. در این سوال بایست بسته به متد داده شده، آرایه q داده شده را به صورت سطری یا ستونی با ماتریس p جمع کنیم. اگر متد row_wise باشد، کافیست q را به کمک تابع np.tile() به تعداد سطرهای p کپی کنیم و سپس جمع بزنیم. اگر متد column_wise باشد بایست q را به تعداد ستون‌های p کپی کنیم و آن را با transpose کردن تبدیل به ستون کنیم و سپس جمع بزنیم.

```
if method == "row-wise":
    extended_q = np.tile(q, (p.shape[0], 1))
    result = p + extended_q
elif method == "column-wise":
  extended_q = np.tile(q, (p.shape[1], 1)).T
  result = p + extended_q
```

۴. در این سوال ابتدا با استفاده از تابع np.random یک ماتریس ۴ * ۴ با مقادیر بین ۱ و ۱۰ می‌سازیم. سپس با استفاده از روش min-max scaling نرمال‌سازی می‌کنیم. به این صورت که ابتدا درایه مینیمم را از همه درایه‌ها کم کرده و سپس بر اختلاف درایه مینیمم و ماکسیمم تقسیم می‌کنیم. درایه‌های ماتریس نهایی بین ۰ و ۱ خواهند بود.

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

۵. ابتدا فایل داده شده را به این صورت می‌خوانیم و ستون‌های آن را جدا می‌کنیم:

```
# read csv file
data_frame = pd.read_csv('data.csv')

# get columns of data_frame
closing_price = data_frame.iloc[:, 1]
dates = data_frame.iloc[:, 0]
```

سپس در بخش اول سوال قیمت هر روز را از روز قبلی کم کرده و در Q5_1 ذخیره می‌کنیم:

```
# Q5_1
Q5_1 = []
for i in range(1, data_frame.shape[0]):
    sub = (closing_price.iloc[i] - closing_price.iloc[i - 1]) / closing_price.iloc[i - 1]
    Q5_1.append(sub)
```

در بخش دوم و سوم سوال با استفاده از توابع آماده numpy میانگین و انحراف معیار مقادیر بخش اول سوال را محاسبه می‌کنیم:

```
# Q5_2
Q5_2 = np.mean(Q5_1)
print(f"average return: {Q5_2}")

# Q5_3
Q5_3 = np.std(Q5_1)
print(f"std_dev return: {Q5_3}")
```

در بخش چهارم سوال با استفاده از matplotlib نمودار قیمت تمام شده بر حسب تاریخ را نشان می‌دهیم. ابتدا ستون Date را به فرمت مناسب درمی‌آوریم و پس از plot کردن برای محورها لیبل‌های مناسب قرار می‌دهیم:

```
# Q5_4
data_frame['Date'] = pd.to_datetime(data_frame['Date'])
plt.figure(figsize=(9, 7))
plt.plot(data_frame['Date'], data_frame['Closing Price'], color='r')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.show()
```

در بخش پنجم سوال نیز مانند بخش ۴ با استفاده از matplotlib این بار نمودار بازده در گذر زمان را نشان می‌دهیم و پس از plot کردن برای محورها لیبل‌های مناسب قرار می‌دهیم:

```
# Q5_5
plt.figure(figsize=(9, 7))
plt.plot(Q5_1);
plt.xticks([])
plt.xlabel('Date')
plt.ylabel('Daily Profit')
plt.show()
```

در بخش ششم، ابتدا با استفاده از توابع argmax, argmin جایگاه بیشترین و کمترین بازده را به دست آورده و سپس تاریخ متناظر آن را پیدا می‌کنیم:

```
# Q5_6
min_randement_idx = np.argmin(Q5_1)
max_randement_idx = np.argmax(Q5_1)

min_randement_date = dates[min_randement_idx + 1]
max_randement_date = dates[max_randement_idx + 1]

print(f'\nmin randement occured on {min_randement_date} with value {Q5_1[min_randement_idx]}')
print(f'max randement occured on {max_randement_date} with value {Q5_1[max_randement_idx]}\n')
```

و در بخش پایانی سوال، ابتدا جایگاه کمترین و بیشترین قیمت تمام شده به به دست می‌آوریم:

```
# Q5_7
min_price_idx = data_frame.iloc[:, 1].idxmin()
max_price_idx = data_frame.iloc[:, 1].idxmax()
```

سپس قیمت و تاریخ متناظر این جایگاه‌ها را پیدا می‌کنیم:

```
min_date = data_frame.iloc[min_price_idx, 0]
min_value = data_frame.iloc[min_price_idx, 1]
max_date = data_frame.iloc[max_price_idx, 0]
max_value = data_frame.iloc[max_price_idx, 1]
```

۶. همانطور که در کد مشخص است در تابع اول، feed forward با استفاده از یک حلقه پیاده‌سازی شده که هر بار مقادیر مربوط به یکی از sample های ورودی را در بردار وزن‌ها ضرب می‌کند اما در تابع دوم مقادیر مربوط به همه sample ها را به یک باره به وسیله تابع آماده np.dot در بردار وزن ضرب می‌کنیم. آنچه قابل توجه است اختلاف زمان اجرای این دو تابع است که همانطور که در زیر مشخص است روش vectorization بسیار سریع‌تر عمل می‌کند.

```
Time spent on calculating the outputs using for loops:
0.0073580741882324222
Time spent on calculating the outputs using vectorization:
0.0012619495391845703
```

۷. در این سوال بایست مقادیر یک آرایه را با مقدار مشخصی مقایسه کنیم و با توجه به بزرگتر یا کوچکتر بودن، در آن جایگاه ۰ و ۱ بگذاریم. من ابتدا با یک مقایسه‌گر خروجی را به شکل بولین درست و غلط درنظر گرفتم و بعد با تابع astype() پایتون آن را به اعداد صحیح ۰ و ۱ مپ کردم:

```
modified_arr = (array > threshold).astype(int)
```

۸. ابتدا در تابع init ماتریس ورودی و ابعاد آن را برای دسترسی ساده‌تر ذخیره کردم. سپس در بخش اول سوال برای چک کردن برابری دو ماتریس ابتدا ابعاد آن‌ها را مقایسه کردم، اگر ابعاد برابر بودند یکی یکی درایه‌ها را چک می‌کنیم و هرجا نابرابری‌ای رخ داد false برمی‌گردانیم:

```
if (self.rows != second_matrix.rows):
    return False
if (self.cols != second_matrix.cols):
    return False

for i in range(self.rows):
    for j in range(self.cols):
        if self.matrix[i][j] != second_matrix.matrix[i][j]:
            return False
return True
```

تست:

```
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 #
print(matrix1.is_equal(matrix2))
```

```
False
```

در بخش دوم ابتدا یک ماتریس خروجی تعریف می‌کنیم سپس درایه ماتریس‌های ورودی را یکی یکی مقایسه می‌کنیم و مقدار ۰ و ۱ در ماتریس خروجی قرار می‌دهیم:

```
result_matrix = [[-1 for _ in range(self.cols)] for _ in range(self.rows)]

for i in range(self.rows):
    for j in range(self.cols):
        if self.matrix[i][j] > second_matrix.matrix[i][j]:
            result_matrix[i][j] = 1
        else:
            result_matrix[i][j] = 0
```

تست:

```
matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = Matrix([[0, 0, 0], [4, 5, 6], [7, 8, 9]])
matrix3 = Matrix([[0, 0, 0], [10, 20, 30], [-1, 8, 10]])

# test proportion of matrices here and show the result #
print(f'{matrix1.is_higher_elementwise(matrix3)}\n')
print(matrix2.is_higher_elementwise(matrix3))
```

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

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

در بخش سوم باید زیر مجموعه بودن دو ماتریس را چک کنیم و یک بولین درست یا غلط بازگردانیم. برای اینکار من ابتدا همه زیرمجموعه‌های ماتریس با ابعاد second_matrix را پیدا کردم و در یک مجموعه به نام subsets ذخیره کردم:

```
subsets = []
subset = [[0 for _ in range(second_matrix.cols)] for _ in range(second_matrix.rows)]

for i in range(self.rows - second_matrix.rows + 1):
    for j in range(self.cols - second_matrix.cols + 1):
        subset = [row[j : j + second_matrix.cols] for row in self.matrix[i : i + second_matrix.rows]]
        subsets.append(subset)
```

سپس چک کردم که آیا second_matrix با هر یک از subsetهای matrix برابر هست یا نه. اگر بود true و اگر با هیچ یک برابر نبود false برگرداندم:

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

تست:

```
matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
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(matrix4))
print(matrix1.is_subset(matrix5))
print(matrix1.is_subset(matrix6))
```

```
True
True
False
```

در بخش آخر سوال بایست ضرب ماتریسی را پیاده‌سازی کنیم. برای اینکار از حلقه‌های تو در تو استفاده کرده‌ام. به این صورت که حلقه اول بر روی ردیف‌های ماتریس اول و حلقه دوم بر روی ستون‌های ماتریس دوم زده می‌شوند. سپس در حلقه سوم هر یک از درایه‌های داخل ردیف ماتریس اول، در درایه‌های داخل ستون ماتریس دوم ضرب می‌شوند و حاصل جمع می‌شود:

```
for i in range(self.rows):
  for j in range(second_matrix.cols):
    for k in range(self.cols):
      result_matrix[i][j] += self.matrix[i][k] * second_matrix.matrix[k][j]
```

تست:

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

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

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