سوال یک

برای پیاده سازی این سوال از تابع های greater,greater_equal,less,less_equal استفاده کردم که کد آن مطابق زیر است:

```
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)
```

سوال دو

در این سوال اگر method برابر با element-wise باشد، آنگاه دو آرایه را در حالت المان به المان در هم ضرب کرده و نتیجه را در متغیر result ذخیره می‌کند. این عملیات به این معنی است که المان اول آرایه array1 با المان اول آرایه array2 ضرب می‌شود، المان دوم با المان دوم ضرب می‌شود و به همین ترتیب تا المان آخر.اگر method برابر با matrix-multiply باشد، آنگاه دو آرایه را در حالت ضرب ماتریسی با یکدیگر ضرب کرده و نتیجه را در متغیر result ذخیره می‌کند. این عملیات به این معنی است که آرایه array1 را با آرایه array2 در حالت ضرب ماتریسی ضرب می‌کند و نتیجه را از طریق عمل ضرب ماتریسی می‌گیرد.به طور خلاصه، این کد بسته به مقدار متغیر method، دو آرایه را در حالت المان به المان یا ضرب ماتریسی با یکدیگر ضرب می‌کند و نتیجه را در متغیر result ذخیره می‌کند.

سوال سه

این کد قسمتی از یک برنامه را نشان می‌دهد که بسته به مقدار متغیر method، عملیات مختلفی را بر روی دو آرایه p و q انجام می‌دهد.اگر method برابر با row-wise باشد، آنگاه دو آرایه p و q با هم جمع می‌شوند. در این حالت، اندازه آرایه‌ها باید یکسان باشد، به این صورت که بتوانند با هم جمع شوند. نتیجه جمع در متغیر result ذخیره می‌شود.اگر method برابر با column-wise باشد، آرایه p با آرایه q در حالت ستون به ستون جمع می‌شود. برای این کار، ابتدا آرایه q با استفاده از [:, np.newaxis] به یک بُعد جدید تبدیل می‌شود تا بتواند با آرایه p جمع شود. سپس جمع دو آرایه در حالت ستون به ستون صورت می‌گیرد و نتیجه در متغیر result ذخیره می‌شود.

سوال چهار

ابتدا، یک آرایه تصادفی به نام xبا ابعاد (شکل) ۴ در ۴ ایجاد می‌شود. اعداد تصادفی در این آرایه در محدوده ۱ تا ۱۰ قرار می‌گیرند.سپس، با استفاده از فرمول نرمال‌سازی، تمام اعداد در آرایه xبه صورت نرمال شده قرار می‌گیرند. در فرمول نرمال‌سازی، هر عنصر از آرایه از مقدار کمینه آرایه تقسیم شده بر انتشار مقدار بزرگترین عنصر تفاوت میان بیشینه و کمینه آرایه است.

سوال پنج

قسمت اول

ابتدا، با استفاده از کتابخانهPandas ، فایل CSV به نام "data.csv" خوانده می‌شود و در یک DataFrame به نام data ذخیره می‌شود.

سپس، با استفاده از ستون Closing Price از داده‌ها، بازده را محاسبه می‌کند. برای محاسبه قیمت پایانی قبلی، از تابع shift(1)استفاده می‌شود تا مقدار قبلی ستون Closing Price را به یک واحد به سمت بالا منتقل کند.در نهایت، نتایج بازده محاسبه شده را نشان می‌دهد.

قسمت دوم

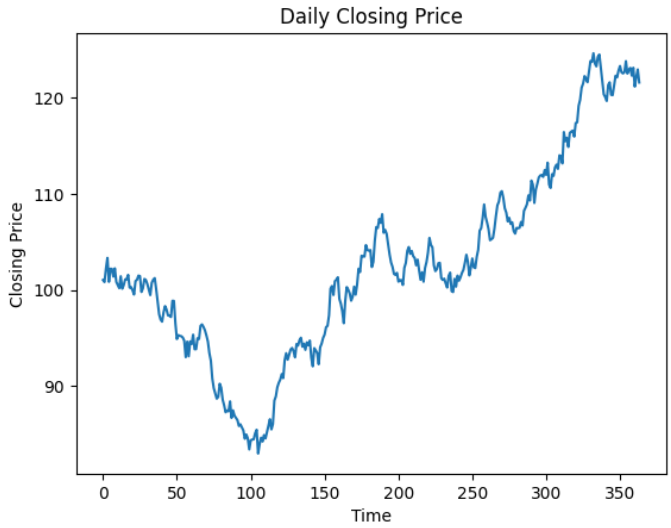
با استفاده از تابع mean میانگین حساب میشود.میانگین برابر است با:-0.0005548260008486608

قسمت سوم

با استفاده از std واریانس را حساب می کنیم.واریانس برابر است با:0.009455978850317194

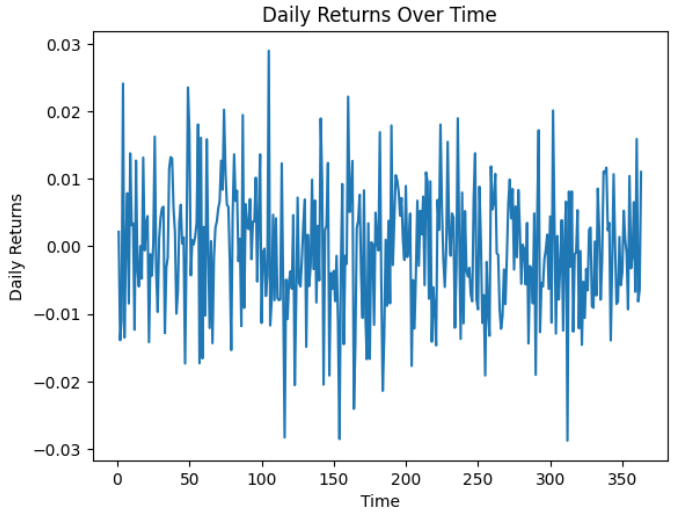
قسمت چهارم

نمودار به صورت زیر است:



قسمت پنجم

نمودار به صورت زیر است:



قسمت ششم

در ابتدا، متغیرهای max_return_day و min_return_day برابر با روزهایی هستند که بازگشت بیشینه و کمینه را دارند، به ترتیب. این اطلاعات با استفاده از تابع idxmax() و idxmin() از سری زمانی returns استخراج می‌شوند.

سپس با استفاده از متغیرهای max_return_day و min_return_day، مقدار بازگشت بیشینه و کمینه از سری زمانی returns در متغیرهای max_return_value و min_return_value ذخیره می‌شوند.

در خطوط چاپ، اطلاعات مربوط به روز و مقدار بازگشت بیشینه و کمینه چاپ می‌شوند. احتمالاً از داده‌های موجود در ستون Date استفاده می‌کند تا تاریخ مربوطه را نشان دهد.

خروجی به صورت زیر میشود:

```
Day with the maximum return: 4/16/2023 Return value: 0.028963574613605738
Day with the minimum return: 11/9/2023 Return value: -0.02878633838810639
```

قسمت هفتم

این قسمت هم از لحاظ کدی مثل قسمت شش هست نتایج به صورت زیر است:

```
Date with the maximum price: 11/29/2023 Price value: 124.6180108
Date with the minimum price: 4/16/2023 Price value: 82.96821012
```

سوال شش

درتابع for_loop_feed_forward یک عملیات feed-forward را با استفاده از یک حلقه for انجام می‌دهد. عملیات feed-forward با استفاده از داده‌های ورودی X و وزن‌ها w انجام می‌شود.

در خطوط اول تابع، تعداد نمونه‌ها را در متغیر num_samples ذخیره می‌کنیم. سپس یک ماتریس خروجی به نام outputs با ابعاد (num_samples, 1) با مقادیر صفر ایجاد می‌شود.

در حلقه اول for، از ۰ تا num_samples حرکت می‌کنیم و در هر مرحله، حلقه دوم for را از ۰ تا تعداد ویژگی‌ها در X اجرا می‌کنیم. در هر مرحله، مقدار outputs[i] با جمع ضرب ویژگی‌ها در وزن‌های متناظر به‌روزرسانی می‌شود.

در تابع vectorized_feed_forward با استفاده از np.dot روش vectorization انجام میدهیم از لحاظ زمانی هم نتایج به صورت زیر است:

```
Time spent on calculating the outputs using for loops:
3.2260282039642334
Time spent on calculating the outputs using vectorization:
0.0010008811950683594
```

تابع vectorized_feed_forward به دلیل استفاده از عملیات ضرب ماتریسی به صورت برداری، در عملیات feed-forward بسیار سریعتر اجرا می‌شود نسبت به تابع for_loop_feed_forward که از حلقه for برای محاسبات استفاده می‌کند.

سوال هفت

در ابتدا، دو متغیر low_value و high_value را تعریف می‌کنیم که به ترتیب مقدار جایگزین برای عناصر کوچکتر از آستانه و برای عناصر بزرگتر از آستانه هستند. در این حالت، مقدار ۰ به عناصر کوچکتر از آستانه و مقدار ۱ به عناصر بزرگتر از آستانه جایگزین می‌شود.

سپس، با استفاده از تابع np.where، آرایه array را با آستانه مقایسه می‌کنیم. اگر عنصری بزرگتر از آستانه باشد، مقدار high_value جایگزین می‌شود؛ در غیر این صورت، مقدار low_value جایگزین می‌شود. نتیجه این عملیات در ماتریس modified_arr ذخیره می‌شود.

در نهایت، ماتریس modified_arr که عناصر بزرگتر از آستانه را با مقدار جایگزین جایگزین کرده است، برگردانده می‌شود.

سوال هشت

این کلاس Matrix یک ساختار داده برای نمایش و عملیات روی ماتریس‌ها را پیاده‌سازی می‌کند. در ادامه، متدهای این کلاس توضیح داده شده‌اند:

- __init__(self, matrix) این متد، ماتریس را به عنوان ورودی دریافت می‌کند و یک شیء از کلاس Matrix را با آن ماتریس ایجاد می‌کند. ورودی matrix یک لیست از لیست‌ها است که ماتریس را نشان می‌دهد. ماتریس ورودی به عنوان ویژگی matrix درون شیء ذخیره می‌شود.
- is_equal(self, second_matrix) این متد ماتریس فعلی را با یک ماتریس دیگر مقایسه می‌کند و بررسی می‌کند که آیا دو ماتریس برابر هستند یا خیر. ورودی second_matrix یک شیء از کلاس Matrix است که ماتریس دیگر را نشان می‌دهد. ابتدا بررسی می‌شود که ابعاد دو ماتریس برابر باشند و سپس هر عنصر را به صورت عنصری مقایسه می‌کند. اگر هر دو ماتریس برابر باشند، True برگردانده می‌شود و در غیر این صورت False.
- is_higher_elementwise(self, second_matrix) این متد مقادیر ماتریس فعلی را با مقادیر ماتریس دیگر به صورت عنصری مقایسه می‌کند و بررسی می‌کند که آیا مقادیر ماتریس فعلی از مقادیر

ماتریس دیگر بزرگتر هستند یا خیر. ورودی second_matrix یک شیء از کلاس Matrix است که ماتریس دیگر را نشان می‌دهد. برای هر عنصر، بررسی می‌شود که آیا مقدار ماتریس فعلی بزرگتر از مقدار ماتریس دیگر است یا خیر. نتیجه به صورت یک ماتریس با همان ابعاد ورودی برگردانده می‌شود.

- is_subset(self, second_matrix) این متد بررسی می‌کند که آیا ماتریس فعلی زیرمجموعه‌ای از ماتریس دیگر است یا خیر. ورودی second_matrix یک شیء از کلاس Matrix است که ماتریس دیگر را نشان می‌دهد. ابتدا بررسی می‌شود که ابعاد ماتریس فعلی کوچکتر یا مساوی ابعاد ماتریس دیگر باشد. سپس ماتریس فعلی را به عنوان یک زیرماتریس در ماتریس دیگر جستجو می‌کند. اگر پیدا شود، بهطوری که تمام عناصر ماتریس فعلی در آن موجود باشند، True برگردانده می‌شود و در غیر این صورت False بر میگرداند.

توضیحاتی در رابطه با نحوه پیاده سازی:

با استفاده از دو حلقه تو در تو، متد یک زیرماتریس از second_matrix با ابعاد مشابه ماتریس فعلی را ایجاد می‌کند. این زیرماتریس با استفاده از حلقه‌ها و با انتخاب قطعه‌های مناسب از second matrix ساخته میشوند.سپس این زیرماتریس در یک نمونه جدید از کلاس Matrix به نام sample قرار داده می‌شود. سپس با استفاده از متد is_equal که برای بررسی برابری دو ماتریس استفاده می‌شود، بررسی می‌شود که آیا ماتریس فعلی و زیرماتریس sample برابر هستند یا خیر.در صورتی که برابری مشاهده شود، به این معنی است که ماتریس فعلی یک زیرمجموعه از second_matrix است و متد True را برمی‌گرداند.در غیر این صورت، بررسی ادامه می‌یابد تا تمام قسمت‌های ممکن از second_matrix بررسی شوند. اگر هیچ زمانی برابری مشاهده نشود، متد False را برمی‌گرداند.

- dot_product(self, second_matrix) این متد ضرب داخلی بین ماتریس فعلی و یک ماتریس دیگر را محاسبه می‌کند. ورودی second_matrix یک شیء از کلاس Matrix است که ماتریس دیگر را نشان می‌دهد. برای محاسبه ضرب داخلی، تمام عناصر ماتریس فعلی با متناظر خود در ماتریس دیگر ضرب می‌شوند و نتیجه به صورت یک ماتریس جدید برگردانده می‌شود.