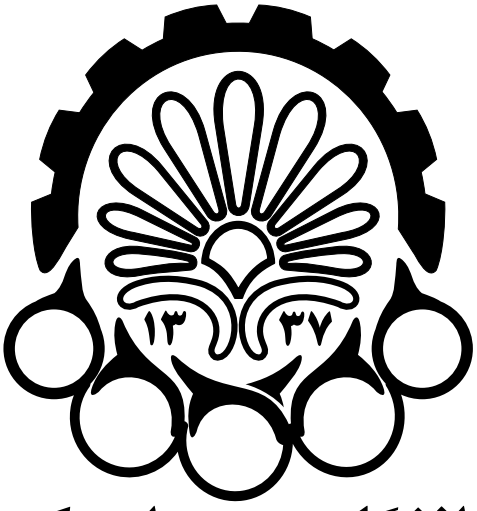


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس : کلان داده

نام و نام خانوادگی : فاطمه توکلی

۴۰۰۱۳۱۰۱۶

تمرین شماره ۰۲

بخش اول : Clustering

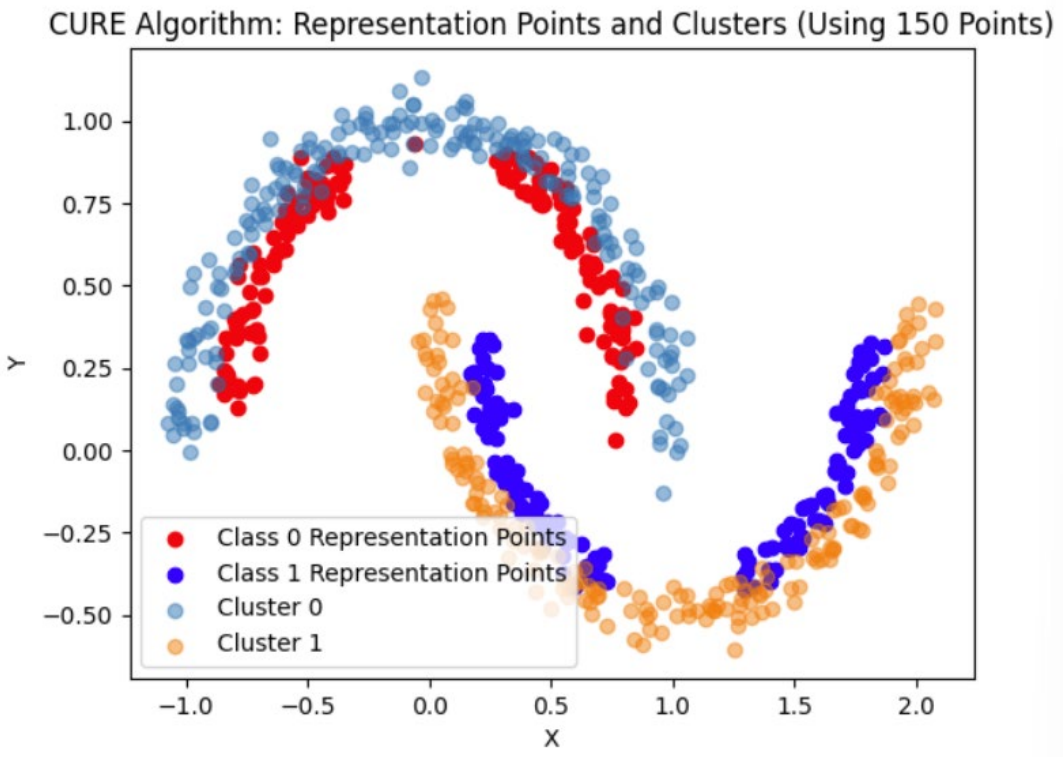
۱-

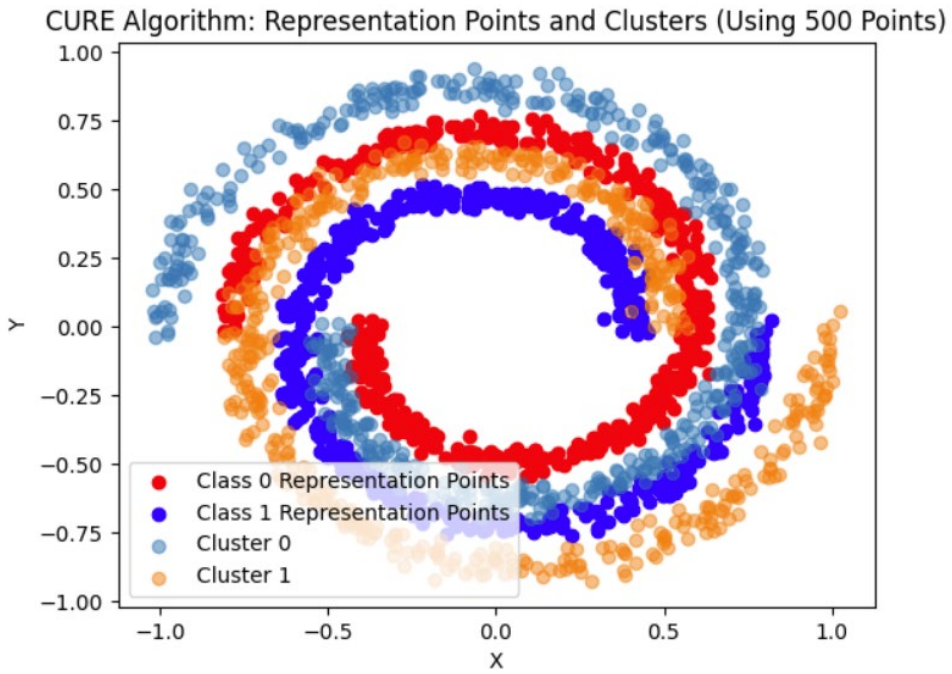
الف) در ابتدا به صورت رندوم از بین نقاط تولید شده تعدادی را انتخاب میکنیم و به عنوان نمونه با این تعداد داده پیش میرویم و سپس بر روی آن ها با استفاده از AgglomerativeClustering دسته بندی سلسله مراتبی روی داده‌ها انجام داده، سپس برای انتخاب نقاط representative در هر دسته به صورت رندوم یک نقطه شروع انتخاب میکنیم و در صورتی که تعداد نقاط representative به حد دلخواه که ورودی تابع است نرسیده باشد به صورت تکرار شونده نقطه بعدی به صورتی انتخاب میشود که بیشترین فاصله اقلیدسی با نقطه فعلی را داشته باشد. و در نهایت تمامی نقاط ۲۰ درصد به سمت مرکز هر دسته شیفت داده میشوند.

در مججموعه داده اول برای اینکه دسته بندی به درستی انجام شود از ۱۰۰۰ داده تولید شده ۴۰۰ داده به عنوان نمونه انتخاب شده و تعداد نماینده ها ۱۵۰ نقطه است

در مجموعه داده دوم ار ۲۰۰۰ داده تولید شده ۱۰۰۰ به عنوان نمونه انتخاب شده و تعداد نماینده ها ۵۰۰ نقطه میباشد.

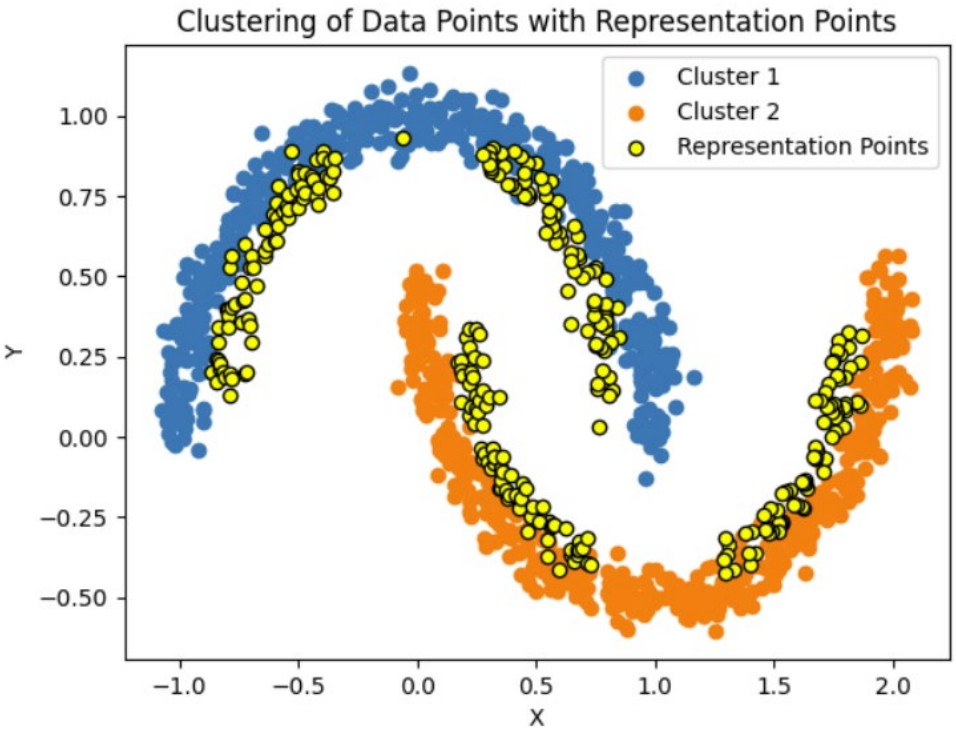
ب) نتیجه برای دو مجموعه داده آورده شده به صورت زیر است:

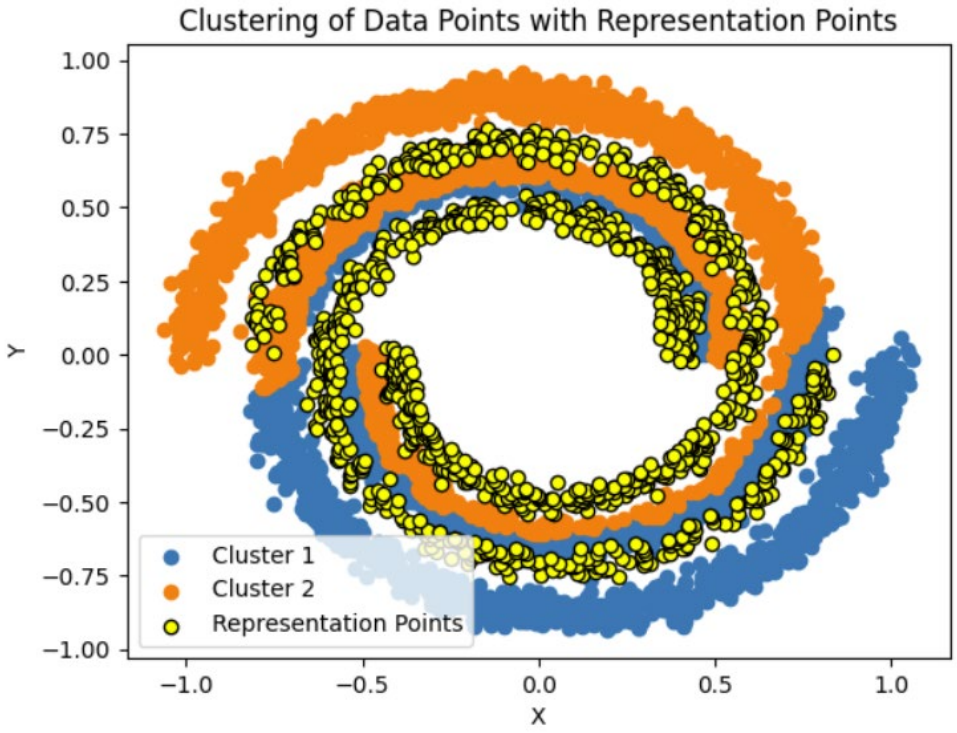




ج) برای دسته بندی نقاط همانطور که در جزوه نیز آورده شده نزدیک نقطه نماینده به هر ورودی را یافته و ورودی به آن دسته اضافه میشود

نتایج برای دو مجموعه داده به صورت زیر است و همانطور که مشاهده میشود برای مجموعه داده spiral به خوبی جواب نداده است





د)

الگوریتم BFR (Boundary-based Fast Representative) یک الگوریتم خوشه‌بندی که به صورت مرحله‌ای عمل می‌کند و در هر مرحله تلاش می‌کند نماینده‌های خوشه‌ها را با استفاده از مرزهای خوشه‌ها بهبود دهد. در ابتدا k دسته در نظر میگیرد و داده را به صورت دسته دسته از حافظه میخواند و هر داده جدید را به یکی از دسته ها نسبت می دهد اگر که فاصله آن ها از یک مقدار استانه کوچکتر باشد. و اگر فاصله از مقدار آستانه کوچکتر نباشد آن را در دسته دیگر به نام compressed sets قرار میدهد و در مرحله بعدی تلاش بر ادغام این compress sets ها از مرحله قبلی دارد و این کار را تکرار میکند تا زمانی که تمام نقاط دسته بندی شوند. کار این الگوریتم خلاصه کردن داده های حجیم است و مسئله انتخاب معیار فاصله و تصمیم گیری راجع به ادغام مجموعه ها بر نتایج اثر دارد

عیب این روش این است که واریانس داده ها را حول محور x یا y محدود میکند و داده باید در جهت یکی از محورها توزیع شده باشد.

مزایای الگوریتم BFR عبارتند از:

۱. کارایی BFR برای پردازش مجموعه داده های بزرگ بهینه شده است و می تواند با سرعت بالا و موازی اجرا شود.

۲. کنترل بر روی ریز خوشه ها BFR می تواند به طور هوشمند ریز خوشه‌ها را تعیین کند و در صورت لزوم تعداد آنها را کاهش دهد تا از بهبود کیفیت خوشه اطمینان حاصل شود.

۳. الگوریتم BFR می تواند با خوشه بندی مجدد خوشه ها، با استفاده از سرعت و کارایی بالای خود، از داده های قبلی زمانی که تغییرات رخ می دهد یا داده های جدید اضافه می شود، استفاده مجدد کند.

مزایای الگوریتم CURE عبارتند از:

۱. CURE در برابر نقاط پرت مقاوم است.

۲. با استفاده از نقاط نماینده برای تعریف خوشه ها، نتایج قابل تفسیر را به راحتی ارائه می‌دهد.

معایب الگوریتم BFR عبارتند از:

۱. BFR به شدت به تخمین دقیق مرزهای خوشه‌ای متکی است، که می‌تواند در مجموعه داده‌های پیچیده با خوشه‌های همپوشانی یا شکل نامنظم چالش برانگیز باشد.

۲. حساسیت به تنظیمات پارامتر: عملکرد BFR می تواند به انتخاب پارامترها مانند تعداد ریز خوشه ها حساس باشد.

معایب الگوریتم CURE عبارتند از:

۱. پیچیدگی محاسباتی: CURE به هزینه محاسباتی بالاتری در مقایسه با BFR نیاز دارد، به خصوص زمانی که با مجموعه داده های بزرگ سروکار داریم، به دلیل نیاز به محاسبه فاصله بین تمام نقاط داده و نمایندگان.

۲. وابستگی به انتخاب نماینده: کیفیت نتایج خوشه بندی در CURE به شدت به انتخاب نقاط نماینده بستگی دارد که می تواند ذهنی باشد و بر عملکرد کلی تأثیر بگذارد.

به طور خلاصه، BFR کارایی و کنترل را بر روی خوشه‌های کوچک ارائه می‌کند، در حالی که CURE استحکام و قابلیت تفسیر را فراهم می‌کند. با این حال، BFR به تخمین مرز و تنظیمات پارامتر حساس است، در حالی که CURE پیچیدگی محاسباتی بالاتری دارد و به انتخاب نماینده وابسته است. انتخاب بین دو الگوریتم به ویژگی های خاص مجموعه داده و اهداف مورد نظر کار خوشه بندی بستگی دارد.

ه) همانطور که دیدیم در مجموعه داده دوم نتوانست اینکار را به درستی انجام دهد یکی از دلایل آن میتواند ۲۰ درصد انتقال به سمت مرکز باشد که باعث شده نقاط نماینده درهم تنیده شوند و معیار فاصله به درستی نتواند دسته‌ها را تفکیک کند.

ی) روش elbow تکنیکی است که برای تعیین تعداد بهینه خوشه ها (k) در الگوریتم خوشه بندی k-means استفاده می شود. شامل ترسیم تعداد خوشه‌ها در برابر مجموع مربعات خطاها (SSE) است.

برای اعمال روش elbow، با انجام خوشه‌بندی k-means با طیفی از مقادیر k، معمولاً از یک مقدار کوچک تا یک مقدار نسبتاً بزرگ شروع می‌شود. برای هر مقدار k، SSE محاسبه شده و بر روی یک نمودار خطی رسم می شود. نمودار معمولاً روند کاهشی در SSE را با افزایش k نشان می‌دهد، زیرا خوشه‌های بیشتر می‌توانند بهتر با نقاط داده تناسب داشته باشند. با این حال، بعد از یک نقطه خاص، کاهش SSE شروع به صاف شدن می کند و شکل خم یا آرنج را در نمودار ایجاد می کند. تعداد بهینه خوشه ها اغلب به عنوان مقدار k در نقطه elbow در نظر گرفته می شود.

با توجه به استفاده از روش elbow برای تعیین k در الگوریتم K-NN، مناسب نیست زیرا الگوریتم K-NN یک الگوریتم طبقه بندی است، نه یک الگوریتم خوشه بندی. الگوریتم K-NN برچسب ها را به نقاط داده جدید بر اساس برچسب های k نزدیکترین همسایه آنها در داده‌های آموزشی اختصاص می دهد. این شامل خوشه بندی یا تعیین تعداد خوشه ها نیست.

در K-NN، مقدار k نشان‌دهنده تعداد همسایه‌هایی است که برای طبقه‌بندی در نظر گرفته می‌شوند، و معمولاً بر اساس دانش قبلی، اعتبارسنجی متقابل یا تخصص دامنه انتخاب می‌شود. ربطی به مفهوم خوشه بندی یا روش آرنجی ندارد. بنابراین، روش elbow برای تعیین مقدار k در الگوریتم K-NN قابل اجرا نیست.

بخش دوم : Bloom Filter

۱-

الف) یکی از دلایلی که این الگوریتم را برای این مسئله مناسب میکند این است که تضمین میکند false negative ندارد در حالیکه از حافظه کمی مصرف می کند و همچنین false positive آن توسط تعداد توابع هش مستقل و تعداد m,n قابل کنترل است و احتمال برخورد را میتوان صفر کرد.

ب) برای به دست آوردن اندازه جدول هش میتوان از فرمول زیر که در جزوه آمده استفاده کرد:

$$\text{false positive probability} = (1 - e^{-km/n})^k$$

که با جایگذاری میتوان به این نتیجه رسید که ۵ برابر جدول اصلی تعریف میشود.

```
# Example usage
primes = [17, 31, 47, 61]  # List of prime numbers

bloom_filter = BloomFilter(1000000, 5000000, len(primes), hash_func_1)

# Load the dataset and add items to the Bloom filter using the specified hash function
dataset_path = "user_dataset.csv"  # Replace with the actual path to your dataset file

with open(dataset_path, "r") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        username = row[0]  # Assuming the username is in the first column
        bloom_filter.add(username)


# Get the bit array
bit_array = bloom_filter.get_bit_array()
```

ابتدا مجموعه داده خوانده شده و لیست اعداد اول تعریف شده به عنوان ورودی n,m, تعداد تابع هش، و همچنین نوع تابع هش را میگیرد و تک تک username هارا خوانده و خانه های جدول هش را با توجه به ۴ تابع هش مقداردهی میکند.

```
def hash_func_1(item, i):
    c = [ord(ch) for ch in item]
    # Select prime number from the list based on the hash function index
    p = primes[i % len(primes)]
    l = len(c)
    min_c = min(c)
    max_c = max(c)
    index = (min_c + math.prod(c)) * p + ((sum(c)) * pow(p, (l // 2))) + (max_c * pow(p, (l - 1)))
    return index
```

تابع هش اول به این صورت تعریف می شود که ابتدا کاراکتر های موجود در رشته را به ارقام تبدیل میکند و سپس از لیست اعداد اول داده شده به ترتیب تابع هش را محاسبه میکند و خروجی را برمیگرداند.

```
index = self.hash_func(item, i) % self.m  # Get the index for each hash function
self.bit_array[index] = True  # Set the corresponding bit to 1
```

خروجی نظیر را بر m باقیمانده میگیرد و در آرایه مقدار آن را برابر با True در نظر میگیرد.

ج) تنها تفاوت در پیاده‌سازی تابع هش خواهد بود و بقیه موارد مانند قبل می باشد:

```
def hash_func_2(item, i):
    c = [ord(ch) for ch in item]
    p = primes[i % len(primes)]
    l = 0
    index = 0
    for x in c :
      index = index + (x *pow(p,l))
      l = l+1
    return index
```

د) برای اینکار username هارا به ترتیب میخوانیم و سپس خروجی تک تک توابع هش را بررسی میکنم اگر تمامی آن ها True بود یعنی آزاد نیست:

```
def test(self, item):
    hash_results = []
    for i in range(self.k):
        index = self.hash_func(item, i) % self.m  # Get the index for each hash function
        hash_results.append(self.bit_array[index])
    if all(hash_results):
        return True
    else:
        print('{} is not in the dataset'.format(item))
        return False
```

و بدین صورت هرکدام در دیتاست نباشد در خروجی می آورد :

```
icolors is not in the dataset
eman88 is not in the dataset
PCthrowsaway is not in the dataset
Anonymous132455 is not in the dataset
koii8 is not in the dataset
netskills001 is not in the dataset
vdubfast01 is not in the dataset
TheSnekMan is not in the dataset
askingmaggie is not in the dataset
```

مقدار accuracy, false positive به صورت زیر است:

```
true positive rate is :  1.0
Accuracy is :  0.87722
```

ه)

```
koii8 is not in the dataset
netskills001 is not in the dataset
mn766132 is not in the dataset
confusedouttamymind is not in the dataset
vdubfast01 is not in the dataset
TheSnekMan is not in the dataset
askingmaggie is not in the dataset
DankFlowers is not in the dataset
WatchWhatHappensLive is not in the dataset
Gamerulf is not in the dataset
deppaotoko is not in the dataset
```

```
true positive rate is :  1.0
Accuracy is :  0.91256
```

ی) خیر این مقدار بهتر است و ممکن است دلایل متفاوتی داشته باشد :

۱- مجموعه داده مورد استفاده ممکن است ویژگی های خاصی داشته باشد که دستیابی به نرخ مثبت کاذب مورد انتظار را برای فیلتر بلوم چالش برانگیز می کند.

۲- توابع هش انتخاب شده ممکن است برای مجموعه داده بهینه نباشد، که منجر به مثبت کاذب بالاتر می شود.

۳- اندازه آرایه بیت یا تعداد توابع هش استفاده شده در فیلتر Bloom ممکن است برای مجموعه داده مناسب نباشد و بر نرخ مثبت کاذب تأثیر بگذارد.

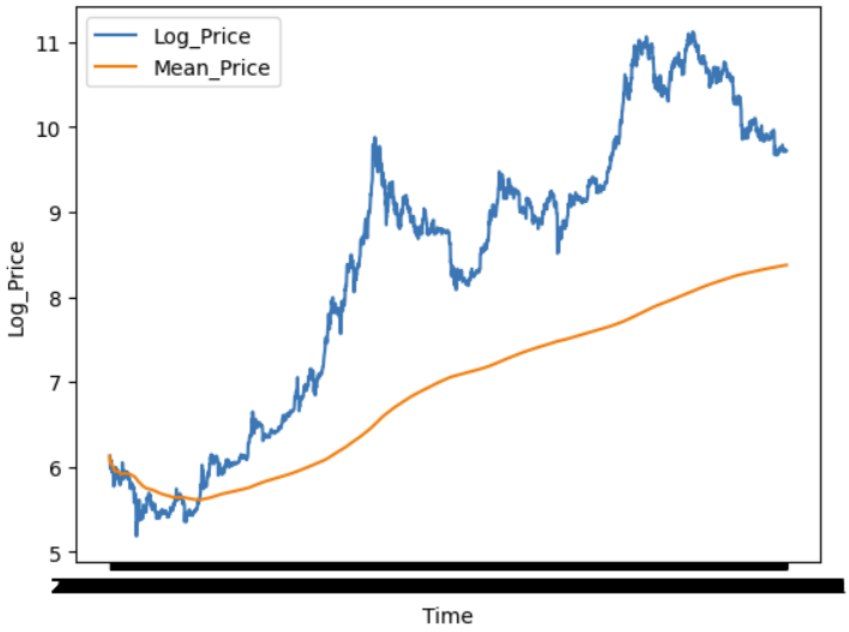
۴- تصادفی بودن مجموعه داده و انتخاب پرس و جو نیز می تواند به تغییرات در نرخ مثبت کاذب کمک کند.

بخش سوم : Data Stream

۱-

الف) بعد از جدا کردن ستون close و همچنین لگاریتم گرفتن از آن ستون، ردیف به ردیف به صورت زیر میانگین را حساب کرده:

```
# Iterate through the log price row by row
for i in range(len(log_close)):
    mean = (mean * i + log_close[i]) / (i + 1)
    mean_values.append(mean)
```

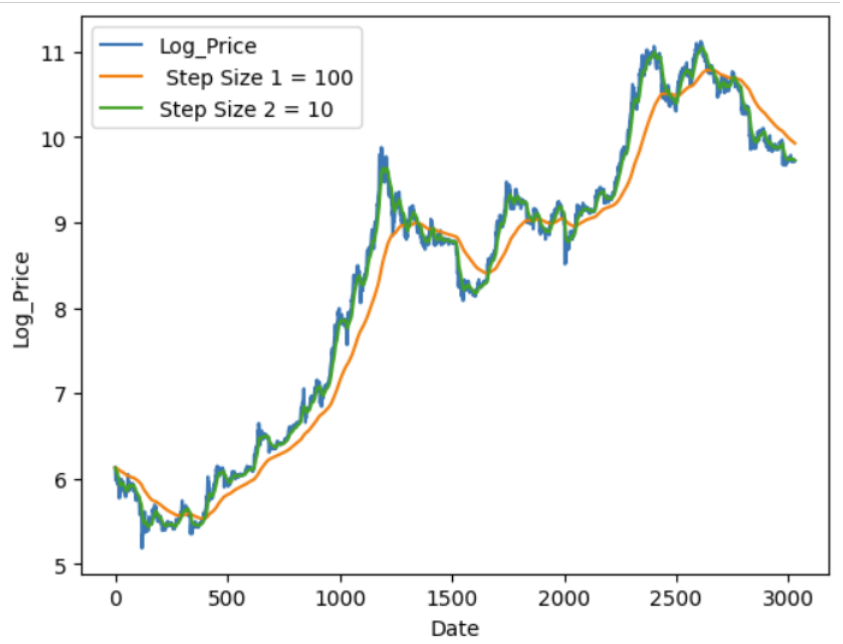


ب) دو مقدار برای step size و تابع لگاریتمی برای آن تعریف میکنیم:

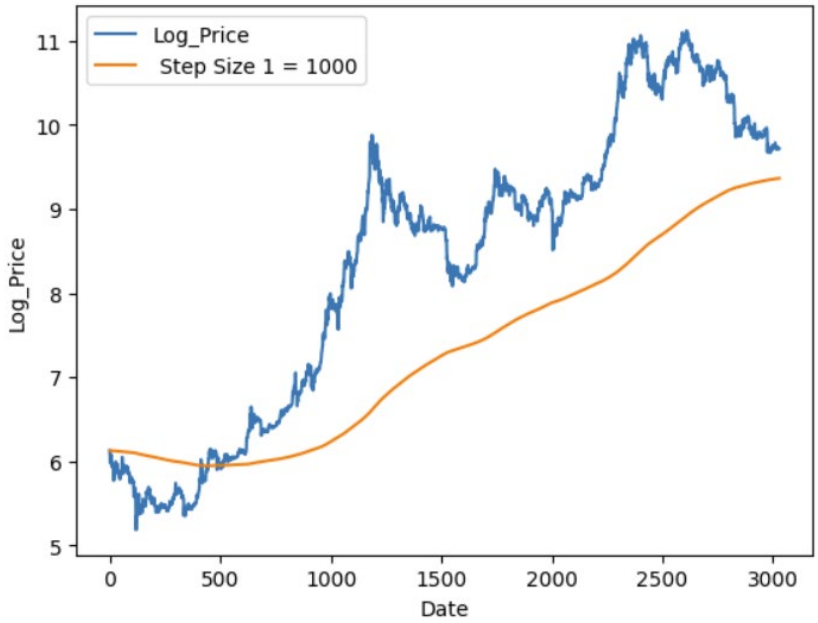
```
stepsize1 = 100
stepsize2 = 10
alpha1 = 1 - math.exp(-1 / stepsize1)
alpha2 = 1 - math.exp(-1 / stepsize2)
```

میانگین وزن دار به صورت زیر تعریف میشود:

```
for i in range(1, len(log_close)):
    new_price = log_close.iloc[i]
    ew_average1 = new_price * alpha1 + (1 - alpha1) * ew_average1
    ew_average_list1.append(ew_average1)
```



اگر بخواهیم شبیه قسمت اول شود باید stepsize =1000 قرار داده شود:



ج) در این قسمت هم از میانگین وزن دار برای محاسبه اختلاف هر داده از واریانس استفاده کردیم هم برای میانگین گیری واریانس ها به صورت وزن دار عمل کردیم، و مقدار آستانه را به این صورت قرار داده که اگر ۴۰ بار به صورت متوالی قدر مطلق اختلاف واریانس از میانگین وزن دار واریانس بیشتر از ۰.۰۱ شد آن نقطه را به عنوان هشدار در نظر گرفتیم:

```
if abs(variance - var_mean) > 0.01:
    count += 1
    if count >= 40:
        points.append((i, new_price))
else:
    count = 0
```

و حاصل به صورت زیر است: